On Automating Hyperparameter Optimization for Deep Learning Applications

N. Shawki[1], R. Rodriguez Nunez[2], I. Obeid[1] and J. Picone[1]

1. Neural Engineering Data Consortium, Temple University, Philadelphia, Pennsylvania, USA
2. Dynamical Systems Lab, Temple University, Philadelphia, Pennsylvania, USA
{nabila.shawki, renato.rodriguez, obeid, picone}@temple.edu

Abstract— Given a large amount of data and appropriate hyperparameters, deep learning techniques can deliver impressive performance if several challenging issues with training, such as vanishing gradients, can be overcome. Often, deep learning training techniques produce suboptimal results because the parameter search space is large and populated with many less-than-ideal solutions. Automatic hyperparameter tuning algorithms, known as autotuners, offer an attractive alternative for automating the training process, though they can be computationally expensive. Additionally, autotuners democratize state-of-the-art machine learning approaches and increase the accessibility of deep learning technology to different scientific communities and novice users. In this paper, we investigate the efficacy of autotuning using Keras Tuner on both synthetic and real-world datasets. We show that autotuning performed well on synthetic datasets but was inadequate on real data. As we increase model complexity, autotuning produces errors that are tedious to resolve for those with limited experience in machine learning. Avoiding overfitting, for example, requires extensive knowledge of an algorithm's unique characteristics (e.g., adding dropout layers). Autotuning tools are excellent for creating baseline models on new datasets, but they need more attention to formulate optimal solutions for end-users with less background in deep learning. Because of this, manual tuning based on domain knowledge and experience is still preferred in machine learning because it produces better performance, even though it requires extensive machine learning expertise.

I. Introduction

In the last decade, there has been a surge in the popularity of deep learning (DL) in established fields such as human language technology [1], image classification [2], and social network analysis [3]. Several factors have contributed to the widespread use of and growing interest in deep learning, including the availability of inexpensive high-speed computation using graphical processing units (GPU) and the development of robust training algorithms that find high-performance solutions [3]. Deep learning offers the potential to avoid manually engineering solutions for specific applications if a large amount of training data is available and if convergence issues during training can be overcome.

Fortunately, numerous datasets with enough data are now available in many fields. Researchers are utilizing multiple deep learning frameworks as well as open-source code to streamline their research methods. The availability of pre-trained models developed by leading research groups has also had a significant impact [3]. The net result of these advances, however, has been an unprecedented interest in applying deep learning to a wide range of problems reminiscent of the interest in artificial intelligence in the 1980's. This has led some researchers and technologists, and many commercial enterprises, to believe that what used to be known as knowledge engineering is no longer needed because the entire technology development process can be automated using inexperienced technologists [4]-[6].

Solving a problem with deep learning often follows a simple technology development pipeline shown in Figure 1. We begin with collecting the data, analyzing and preparing the data, feature engineering (if necessary), selecting or designing a model, training a model with hyperparameter tuning, validating the model with an open set test, and finally, deploying the model [7]. Hyperparameter tuning is a crucial step to obtain a near-optimal solution. Hyperparameters can be divided into two categories: training-related (e.g., learning rate, batch size, dropout rate, and epoch count) and model design-related (e.g., model structure, regularizers, and activation functions). However, due to the number of hyperparameters involved, it is nearly impossible to explore all possible combinations [8][9].

The general approach to tuning hyperparameters involves manual search in which experiments are conducted by an expert based on prior knowledge, experience, and, of course, trial-and-error. This process is often time-consuming, tedious, and computationally expensive since it requires a detailed understanding of the algorithms. Autotuning is an active research area that involves automated search techniques to find an optimal solution. There are approaches to automate the optimization process, including Grid Search, Random

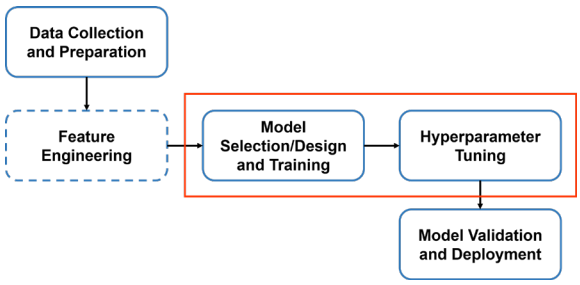

Figure 1. The technology development pipeline

Search, Bayesian Optimization, and Gradient-based Optimization [10][11]. The appeal of autotuning techniques lies not only in the fact that it reduces the trial-and-error period but also provides an off-the-shelf aid for experts in other fields with limited knowledge in machine learning [12]. In this paper, we investigate if automatic tools for hyperparameter tuning are sufficient to achieve generalized solutions on challenging real-world problems, such as automatic interpretation of electroencephalography (EEG) signals and digital pathology whole slide images (WSI). We investigate tuning hyperparameters both manually and automatically using the Keras Tuner and show that autotuning by itself is not a sufficient solution. We introduce several synthetic data sets that were designed to challenge the generalization capability of a machine learning system.

II. Hyperparameter Tuning

The process of selecting hyperparameters is a complex optimization problem. In practice, this is often tackled by grid searches that follow some sort of gradient descent strategy. These approaches are common because they are simple to implement and parallelize, and they often provide insight into the hyperparameter optimization surface. However, ongoing research suggests that automated random search optimization is a more efficient alternative that often yields as good or better models than manual methods due to their ability to search larger configuration spaces [9]-[15]. We will discuss how manual and random search methods are used to optimize the hyperparameter selection process using the Keras Tuner Toolkit [16].

The goal of a typical learning algorithm $\mathcal{A}$ is to map a finite set of samples $X^{(train)}$, obtained from a ground truth distribution, $\mathcal{G}_x$, to a function f that minimizes some expected loss $\mathcal{L}(x: f)$. The learning algorithm $\mathcal{A}$ is defined by its selection of hyperparameters λ that control its architecture and the training (learning) process. Following the definition of the learning algorithm $\mathcal{A}_\lambda$, via its hyperparameters λ, the algorithm can produce f through optimization of a learning criterion with respect to a set of parameters θ. From this representation of $\mathcal{A}_\lambda$, it can be concluded that the actual learning algorithm is only obtained after selection of the hyperparameters λ, and that the optimality of f is dependent on the selection of optimal λ for a given training set $X^{(train)}$:

$$f = \mathcal{A}_\lambda\left(X^{(train)}\right). \tag{1}$$

Thus, to find the optimal f that minimizes some expected loss $\mathcal{L}(x: f)$, we must choose λ such that the generalization error given by, $\mathbb{E}_{x\sim\mathcal{G}_x}\left[\mathcal{L}\left(x: \mathcal{A}_\lambda\left(X^{(train)}\right)\right)\right]$, is minimized. By minimizing the expected generalization error with respect to the hyperparameters λ over the search space Λ, we obtain:

$$\lambda^{(*)} = argmin_{\lambda\epsilon\Lambda}\, \mathbb{E}_{x\sim\mathcal{G}_x}\left[\mathcal{L}\left(x: \mathcal{A}_\lambda\left(X^{(train)}\right)\right)\right]. \tag{2}$$

Next, using cross-validation, the optimization problem $\lambda^{(*)}$ can be expressed in terms of the hyperparameter response function Ψ:

$$\lambda^{(*)} = argmin_{\lambda\epsilon\Lambda}\Psi(\lambda) \equiv \tilde{\lambda}. \tag{3}$$

In traditional manual tuning methods, the optimal set of hyperparameters $\tilde{\lambda}$ is found by (manually) defining a set of n trail points $\{\lambda^{(1)}, \dots, \lambda^{(n)}\}$, evaluating the response surface $\Psi(\lambda)$ on each of the n points, and selecting the best performing hyperparameter set as $\tilde{\lambda}$. This approach helps gain intuition into the response surface and helps identify promising regions in the search space Λ. However, to find the optimal solution $\tilde{\lambda}$, it is often required that researchers have domain expertise on the given data $X^{(train)}$, and experience working with machine learning techniques [10].

Autotuning methods such as the Keras Tuner tool were introduced to facilitate optimization by providing a user-friendly platform for the automated search of optimal hyperparameter combinations. They achieve this by first defining a hyperparameter search space Λ that is broad enough to include all reasonable combinations of λ. Then they require the development of a hypermodel of the selected deep learning method that supports variation of the hyperparameters, including ones that can change the architecture of the chosen method. Once both requirements are met, the random search tool (Keras Tuner) draws n independent hyperparameter sets $\{\lambda^{(1)}, \dots, \lambda^{(n)}\}$ from a uniform density from the search space Λ. Then, similarly to the manual tuning method, the response surface $\Psi(\lambda)$ is evaluated on each of the randomly drawn hyperparameter sets. Finally, the best performing set of hyperparameters is selected as the optimal solution $\tilde{\lambda}$ [9].

III. Experimental Data

To investigate the generalization capability of the autotuning process, we used five balanced datasets of varying difficulties. Table 1 provides an overview of the datasets. Three of these datasets were created manually

Table 1. Description of the datasets

#	Description	No. Classes	Train	Dev	Eval
08	Synthetic data	3	300,000	15,000	15,000
09	Synthetic data	2	500,000	250,000	250,000
10	Synthetic data	2	100,000	10,000	10,000
11	Cancer Images	2	10,000	2,500	2,500
12	EEG Signals	2	10,000	2,500	2,500

to simulate complex nonlinear decision surfaces. The remainder were extracted from real-world data. Each of the datasets were partitioned into three standard subsets: 'train' for training the model, 'dev' for validation and tuning, and 'eval' for blind evaluation. The datasets are part of a series of datasets for a graduate level machine learning course, Introduction to Machine Learning and Pattern Recognition, and are publicly available at: *https://www.isip.piconepress.com/courses/temple/ece_8527/resources/data/*.

The three synthetic datasets were created manually to test the generalization capability of a machine learning system. These sets were created using an open-source Python-based machine learning tool called IMLD that we have developed [17]. Dataset #08 contains three classes with complex distributions requiring a model to find an optimal non-linear decision surface. Dataset #09 was generated using five tight equally-spaced Gaussian distributions. The samples in dataset #10 were manually created as well using IMLD and augmented using Gaussian white noise. The distributions for the subsets in dataset #10 are shown in Figure 2. Each sample in these datasets (#08-10) contains two features so that they are easy to visualize using plotting tools available in Python and MATLAB.

Two additional datasets based on real-world data were also created. Dataset #11 comprises images containing two types of breast cancer data. The images were collected from the Temple University Digital Pathology (TUDP) database, which contains whole slide images (WSI) of cancerous and non-cancerous tissues. The details of this dataset can be found in [18]. The annotation and release of the breast cancer slides is an ongoing project. The breast cancer slides in TUDP can contain two types of cancer as shown in Figure 3: ductal carcinoma in situ (dcis) and invasive ductal carcinoma in situ (indc). Based on the annotations, we traversed the slides and collected patches of 512×512 pixels. We then randomly selected patches to generate dataset #11. The WSIs for each individual set were selected before the patch extraction so that overlaps among training, validation, and testing are avoided.

Lastly, dataset #12 contains time series data collected from electroencephalograph (EEG) records available in the Temple University Hospital EEG Seizure Corpus

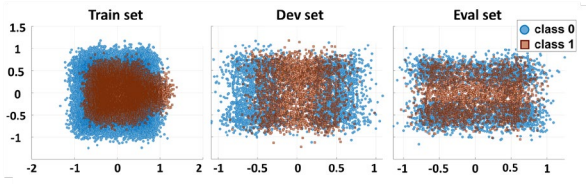

Figure 2. The train, dev, and eval sets for dataset #10

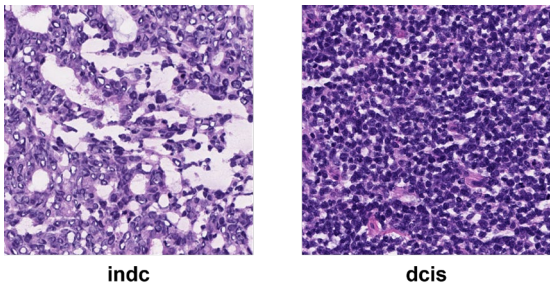

Figure 3. Cancer image samples from dataset #11.

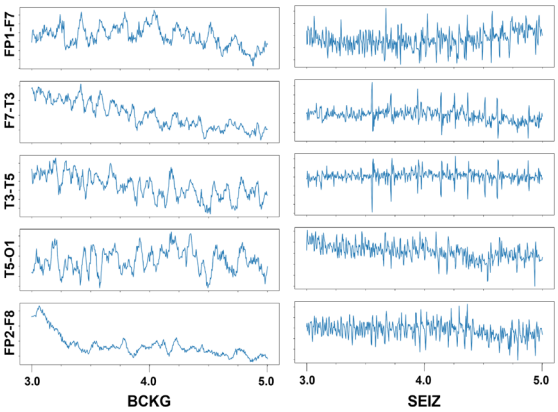

Figure 4. Two-second signals from dataset #12

(TUSZ). A detailed description of the data, channels, the file formats, and processing can be found in [19]. Seizure detection is a very challenging problem because the signals are noisy and contain numerous artifacts [19]. The files are annotated with two classes which are seizure (SEIZ) and background (BCKG). We selected 20 channels from the EEG records and separated the segments that were more than 15 seconds long. Next, segments that were 10-seconds long were extracted from these by avoiding overlaps in the longer segments. Figure 4 shows five channels from typical 2-second, 20-channel background and seizure samples.

The experiments were conducted using a mixture of CPUs and GPUs depending on the complexity of the data. Google Colab [20] was used for the first three datasets for autotuning experiments (using an Intel Xeon 2.30 Hz processor and NVIDIA K80 and T4 GPUs). The manual tuning experiments for those datasets were executed on AMD Opteron 2.40 GHz and Intel Xeon 2.20 GHz processors. The experiments on the EEG and cancer datasets for both manual and autotuning were performed on GPUs (NVIDIA RTX 2080).

IV. Experimental Design

We used PyTorch for the manual tuning experiments. For creating non-deep learning baselines, we used k-nearest neighbors (KNN) and random forests (RNF) [21], both of which are known to be quite robust across a wide range of applications. Using these algorithms, we also

established the Bayes error rate (BER) [22] on datasets #08-10. No preprocessing was necessary for the synthetic datasets. We tuned multilayer perceptron (MLP) models for the synthetic datasets since they have neither temporal nor spatial context. For the cancer image sets, we designed a ConvNet [23] and fed the training images by randomly cropping, jittering, and rotating. We also normalized the image before feeding it to both manually tuned and autotuned models.

Since dataset #12 contains sequential data, a hybrid model with a ConvNet [23] and a Long Short-term Memory Network (LSTM) [24] was designed. For both manual and autotuning experiments, we preprocessed the data using a bandpass filter in the range $0.5 - 35$ Hz and then reduced the sampling rate to 50 Hz. We then applied framing and windowing to extract 0.1-second frames with 0.3-second windows.

We used TensorFlow and the Keras Tuner Toolkit for the automated tuning experiments. The setup process begins with the definition of the hyperparameter search space Λ. For the Keras Tuner, Λ is defined by methods of the HyperParameters (hp) class, such as the Choice and Int methods. These methods are used to provide hyperparameters choices in the form of: (a) lists of strings (e.g., for iterating through different activations functions) and (b) inclusive ranges of integers (e.g., for probing different numbers of hidden layers or neurons). Along with the search space Λ, a hypermodel is developed to better encapsulate the search space. The hypermodel implements a build (self, hp) method that takes the hp (set of hyperparameters λ) argument to create a Keras model instance. As the tuner iterates through the hyperparameter sets randomly drawn from Λ, the hypermodel creates trainable Keras models that implement the architecture (e.g., number of hidden layers) and learning process (e.g., optimizer) defined in each λ. A description of the arguments used in the Keras Tuner Toolkit are shown in Table 2.

MLP models were tuned for datasets #08-10. For the automated tuning of these models, the Int and Choice methods were used to define the search space Λ, which included the following search dimensions: (a) the number of hidden layers in the model, (b) the number of neurons per hidden layer, (c) the activation function used per layer (i.e. the input, hidden, and output layers), (d) the dropout rate, (e) the loss function, (f) the optimizer, and (g) the metric. For the model developed for dataset #12 which is an LSTM, the search space resembles that of the MLP models. For the ConvNet model developed for dataset #11, the search space Λ also included: (h) the number of output filters in the convolution, (i) the kernel size, and (j) the size of the pooling window. Lastly, the hypermodels were developed using the standard build methods for sequential MLP, ConvNet, and LSTM.

Table 2. Description of arguments in the Keras Tuner

Name	Argument	Type	Values
hypermodel (hm)	HyperModel instance	class	–
hyperparameters (hp)	HyperParam instance	class	hp.Int hp.Choice
Objective	optimization metric	String	val_accuracy
max_trials	number of model configs.	Int	500
Seed	random seed	Int	42
max_epochs	epochs to train one model	Int	10 to 35
batch_size	samples per gradient update	Int	8 to 512
allow_new_entries	allow hm to request hp not in Λ	BOOL	True
tune_new_entries	add hp requested by hm to Λ	BOOL	True

V. Results

Our performance comparison metric was error rate since all datasets were balanced. We compared the model complexity, time to achieve an optimal solution, and implementation difficulties involved with the frameworks. It should be mentioned that all techniques were only given one chance at decoding the eval set.

Table 3 displays the performance of different models on the datasets (column DS). MLP-M denotes manually tuned models while MLP-A stands for autotuned models. For dataset #10, we tuned models with and without dropout to observe the autotuner's behavior. In this case, the models appended with a "1" refer to the models without dropout, while the models appended with a "2" refer to models with dropout. All models for the synthetic datasets, regardless of the tuning method, used an Adam optimizer. Cross-entropy loss was applied during manual tuning whereas the autotuner used Sparse Categorical Cross-entropy loss. For dataset #10, the manually tuned models used a learning rate (LR) of 0.0001 with a batch size (bsize) of 16 and trained for 25 epochs. For the auto tuner, these values were 0.001, 512, and 10 respectively.

In Table 3, we provide an experimental estimate of the Bayes Error Rate (BER) [22]. This is an estimate of the lowest achievable error rate for that set that was calculated by doing closed-loop training with KNN and RNF for the train, dev, and eval sets independently. In all cases, the error rates for the manually tuned models, which are highlighted by the shaded cells in Table 3, were lower than those for the autotuned models. The time to reach the optimal solution for the synthetic datasets

Table 3. Performance of the models (% error rate)

DS	System	Train	Dev	Eval
#08	KNN	23.48	26.62	64.18
	RNF	29.23	29.45	59.77
	MLP-M	32.56	32.29	**56.39**
	MLP-A	36.42	30.70	57.38
	BER	23.48	24.45	20.07
#09	KNN	2.11	3.81	16.63
	RNF	2.06	3.82	18.32
	MLP-M	2.21	3.91	**12.87**
	MLP-A	3.70	5.30	14.22
	BER	2.06	2.30	1.85
#10	KNN	7.63	38.83	33.44
	RNF	2.15	39.74	33.28
	MLP-M1	8.74	38.52	32.80
	MLP-M2	9.91	40.88	**32.00**
	MLP-A1	28.95	26.95	42.07
	MLP-A2	12.42	40.11	32.34
	BER	2.15	11.70	13.74
#11	Manual	12.17	12.56	**20.88**
	Auto	35.28	24.28	41.55
#12	Manual	24.85	27.32	**36.08**
	Auto	50.00	50.00	50.00

(first three datasets) was approximately three hours. Within this time, manual tuning completed 15 manual iterations whereas the autotuner performed 500 explorations of the search space Λ. As the model complexity increased for datasets #11 and #12, the time to achieve the optimal hyperparameter set also increased accordingly.

The model architectures for dataset #10 are shown in Table 4. The decision surfaces generated by these models are shown in Figure 5. The manually tuned MLP architectures are simpler compared to the autotuned MLP models. This tendency was also observed for the manual and auto ConvNets developed for dataset #11 as shown in Table 5. This reduction in model complexity suggests that the manually tuned models are less prone to overfitting and will better generalize to unseen data. This claim is substantiated by the ConvNet and LSTM evaluation results detailed in Table 3, which show that the more complex autotuned models perform poorly on the evaluation set while the manually tuned model performs well on both the training and evaluation sets.

It is important to point out that the performance of the autotuned ConvNet model was limited by a "shallow" exploration of the hyperparameter search space Λ. This limitation was brought upon by the random draw of

Table 4. The model architectures for dataset #10

Layout	MLP-M1 (PyTorch)	MLP-M1 (PyTorch)	MLP-A1 (Keras)	MLP-A2 (Keras)
Input Layer	Linear (2, 64), ReLU	Linear (2, 64), ReLU, Dropout (0.3)	Dense (52), sigmoid	Dense (97), tanh, dropout (0.25)
Layer 1	Linear (64, 32), ReLU,	Linear (64, 32), ReLU, dropout (0.3)	Dense (92), sigmoid	Dense (47), sigmoid, dropout (0.25)
Layer 2	Linear (32, 16), ReLU,	–	Dense (62), sigmoid	Dense (62), exponential, dropout (0.25)
Layer 3	–	–	Dense (72), sigmoid	Dense (37), tanh, dropout (0.25)
Layer 4	–	–	Dense (67), tanh	–
Output Layer	Linear (16,2)	Linear (32,2)	Dense (2), softmax	Dense (2), sigmoid

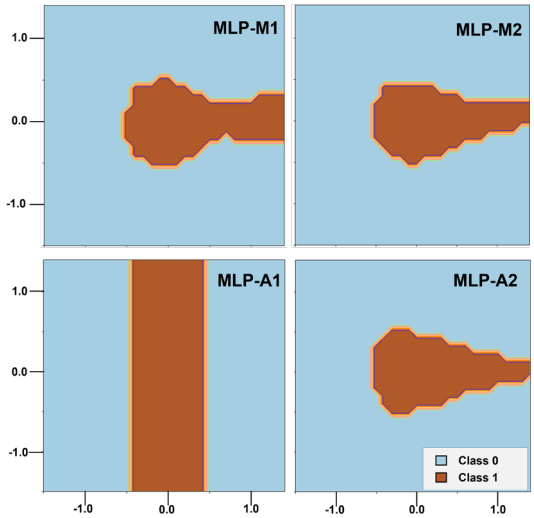

Figure 5. The decision surfaces for dataset #10

"bad" hyperparameter sets λ that caused errors (e.g., negative dimension of the layer input shape) which resulted in a halt of the automatic tunning process, as it currently does not have the capability to skip problematic λ. Similar issues also arose during LSTM training. The manually tuned model could explore a hybrid model while the autotuned model proved difficult to tune only

Table 5. The model architectures for dataset #11

Layout	Manual (PyTorch)	Auto (Keras)
Input Layer	Conv2d (3, 16, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,8), tanh, MaxPool2d (5,5)
Layer 1	Conv2d (16, 32, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,5), tanh, MaxPool2d (3,3) Dropout (0.25)
Layer 2	Conv2d (32, 64, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,5), tanh MaxPool2d (3,3) Dropout (0.25)
Layer 3	Linear (9216, 128), ReLU, dropout (0.25)	Dense (25), ReLU
Layer 4	-	Dense (25), ReLU
Layer 5	-	Dense (25), ReLU
Output Layer	Linear (128, 2)	Dense (2), softmax
Optimizer	SGD	Adam
LR	0.001	0.001
Epoch	15	10
bsize	8	32

with LSTM layers as demonstrated in Table 3. Thus, although the manually tuned models performed well on the complicated EEG and digital pathology image datasets, it is possible that similar performance can be achieved by the autotuned models if a thorough exploration of Λ is completed.

Again, as seen from the decision surfaces in Figure 5, the autotuned model without dropout, MLP-A1, overfitted the dev set while the manually tuned model, MLP-M1, produced a more generalized solution. The autotuner aggressively reduced the error on the validation set, which caused a dip in performance on the eval set. During manual tuning, this was avoided as the performance of the model was being carefully observed on train and dev sets. The decision surfaces were also being carefully reviewed after each change to ensure that the model would not overfit the train or dev sets. This is another example of how ML expertise can significantly enhance the training process.

VI. Conclusions

This paper is an initial investigation of automatic hyperparameter tuning using the Keras Tuner toolkit. Autotuners were developed with the goal of democratizing state-of-the-art machine learning approaches and increasing their accessibility to different scientific communities. However, our observations suggest that limited knowledge on deep learning processes can lead to degraded performance due to the use of incomplete hyperparameter search spaces Λ, such as the example shown for dataset #10. Further, we observed that the efficacy of the tool decreased as the model complexity increased because it yielded errors that are difficult to solve for researchers with limited experience in deep learning. Future work will involves experimenting with more autotuning tools such as Ray Tune [25], Optuna [26], Google Vizier [27], and Microsoft Neural Network Intelligence (NNI) [28] to provide a better analysis of autotuning techniques.

Acknowledgments

This material is supported by the National Science Foundation under grants nos. CNS-1726188 and 1925494. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Open-source libraries that were used to develop the IMLD are: NumPy v1.15.4, PyQT5 v5.13.1, SkLearn v0.20.0, Scipy v1.1.0, and Matplotlib v3.0.2.

References

[1] T. Ma, "How Has Deep Learning Revolutionized Human Language Technology?," in *Proceedings of the IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2020, pp. 1-2. *https://ieeexplore.ieee.org/document/9353633*.

[2] M. D. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A Comprehensive Survey of Deep Learning for Image Captioning," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1-36, Feb. 2019. *https://dl.acm.org/doi/abs/10.1145/3295748*.

[3] S. Pouyanfar *et al.*, "A Survey on Deep Learning: Algorithms, Techniques, and Applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1-36, Sep. 2018. *https://dl.acm.org/doi/abs/10.1145/3295748*.

[4] GreeekDataGuy, "My Advice To Machine Learning Newbies After 3 Years In The Game," *Towards Data Science*, 2021. [Online]. Available: *https://towardsdatascience.com/my-advice-to-machine-learning-newbies-after-3-years-in-the-game-6eef381f540*. [Accessed: 13-Oct-2021].

[5] Splunk.com. (2021). *5 Big Myths of AI and Machine Learning Debunked*. *https://www.splunk.com/en_us/form/5-big-myths-of-ai-and-machine-learning-debunked.html*. [Accessed: 13-Oct-2021].

[6] C. Dossman, "Top 6 Errors Novice Machine Learning Engineers Make," *AI: Theory, Practice, Business*, 2021. [Online]. Available: *https://medium.com/ai³-theory-practice-business/top-6-errors-novice-machine-learning-engineers-make-e82273d394db*. [Accessed: 13-Oct-2021].

[7] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artif. Intell. Med.,* vol. 104, p. 101822, 2020. *https://dl.acm.org/doi/10.1145/3295748*.

[8] H. Cui, G. R. Ganger, and P. B. Gibbons, "MLtuner: System Support for Automatic Machine Learning Tuning." 2018. *https://arxiv.org/pdf/1803.07445.pdf*.

[9] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *J. Mach. Learn. Res.,* vol. 13, pp. 281-305, Feb. 2012. *https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a*.

[10] M. Feurer and F. Hutter, "Hyperparameter Optimization," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer International Publishing, 2019, pp. 3-33. *https://link.springer.*

com/chapter/10.1007/978-3-030-05318-5_1.

[11] Z. Wang, M. Agung, R. Egawa, R. Suda, and H. Takizawa, "Automatic Hyperparameter Tuning of Machine Learning Models under Time Constraints," in *Proceedings of 2018 IEEE International Conference on Big Data (Big Data),* 2018, pp. 4967-4973. *https://ieeexplore.ieee.org/document/8622384*.

[12] D. Yogatama and G. Mann, "Efficient Transfer Learning Method for Automatic Hyperparameter Tuning," in *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014. *http://proceedings.mlr.press/v33/yogatama14.pdf*.

[13] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi, "Sherpa: Robust hyperparameter optimization for machine learning," *SoftwareX*, vol. 12, p. 100591, 2020. *https://www.sciencedirect.com/ science/article/pii/S2352711020303046*.

[14] H. Choi and S. Park, "A Survey of Machine Learning-Based System Performance Optimization Techniques," *Appl. Sci.*, vol. 11, no. 7, 2021. *https://www.mdpi.com/2076-3417/11/7/3235*.

[15] A. F. Rogachev and E. V. Melikhova, "Automation of the process of selecting hyperparameters for artificial neural networks for processing retrospective text information," in *IOP Conference Series: Earth and Environmental Science*, 2020, vol. 577, p. 12012. *https://iopscience.iop.org/article/10.1088/1755-1315/577/1/012012/pdf*.

[16] T. O'Malley et al., "Keras Tuner," 2019. [Online]. Available: *https://keras.io/keras_tuner*. [Accessed: 29-Jul-2021].

[17] T. Cap, A. Kreitzer, M. Miranda, D. Vadimsky, and J. Picone, "IMLD: A Python-Based Interactive Machine Learning Demonstration," in *Proceedings of the IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2021, pp. 1-4. *https://www.isip.piconepress.com/publications/conference_presentations/2021/ieee_spmb/imld/*.

[18] N. Shawki et al., "The Temple University Digital Pathology Corpus," in *Signal Processing in Medicine and Biology: Emerging Trends in Research and Applications*, 1st ed., I. Obeid, I. Selesnick, and J. Picone, Eds. New York City, New York, USA: Springer, 2020, pp. 67-104. *https://www.springer.com/gp/book/9783030368432*.

[19] V. Shah et al., "The Temple University Hospital Seizure Detection Corpus," *Front. Neuroinform.,* vol. 12, pp. 1-6, 2018. *https://www.frontiersin.org/articles/10.3389/fninf.2018.00083/full*.

[20] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications," *IEEE Access*, vol. 6, pp. 61677-61685, 2018. https://ieeexplore.ieee.org/document/8485684.

[21] P. Thanh Noi and M. Kappas, "Comparison of Random Forest, k-Nearest Neighbor, and Support Vector Machine Classifiers for Land Cover Classification Using Sentinel-2 Imagery," *Sensors*, vol. 18, no. 1, 2018. *https://www.mdpi.com/1424-8220/18/1/18*.

[22] K. Tumer and J. Ghosh, "Estimating the Bayes error rate through classifier combining," in *Proceedings of 13th International Conference on Pattern Recognition*, 1996, vol. 2, pp. 695-699. *https://ieeexplore.ieee.org/document/546912*.

[23] Y. LeCun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time Series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, Massachusetts, USA: MIT Press, 1998, pp. 255-258. *https://dl.acm.org/doi/10.5555/303568.303704*.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory.," *Neural Comput.*, vol. 9, no. 8, pp. 1735-80, 1997. *https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory*.

[25] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A Research Platform for Distributed Model Selection and Training." 2018. *https://arxiv.org/pdf/1807.05118.pdf*.

[26] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 2623-2631. *https://dl.acm.org/doi/10.1145/3292500.3330701*.

[27] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487-1495. *https://dl.acm.org/doi/10.1145/3292500.3330701*.

[28] S. Raschka, J. Patterson, and C. Nolet, "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," *Information*, vol. 11, no. 4, 2020. *https://www.mdpi.com/2078-2489/11/4/193*.