BiLSTM-CNN Hyperparameter Optimization for Speech Emotion and Stress Recognition

Agustinus Bimo Gumelar
Dept. of Electrical Engineering, Faculty of Intelligent Electrical and Information Technology (ELECTICS), Institut Teknologi Sepuluh Nopember,
Surabaya, Indonesia
bimo.19071@mhs.its.ac.id
bimogumelar@ieee.org

Eko Mulyanto Yuniarno
Dept. of Electrical Engineering, Dept. of Computer Engineering, Faculty of Intelligent Electrical and Information Technology (ELECTICS), Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
ekomulyanto@ee.its.ac.id

Derry Pramono Adi
Dept. of Electrical Engineering, Faculty of Intelligent Electrical and Information Technology (ELECTICS), Institut Teknologi Sepuluh Nopember,
Surabaya, Indonesia
derryalbertus@ieee.org

Adri Gabriel Sooai
Dept. of Computer Science, UNIKA Widya Mandira,
Kupang, NTT, Indonesia
adrigabriel@unwira.ac.id

Indar Sugiarto
Dept. of Electrical Engineering, Petra Christian University,
Surabaya, Indonesia
indi@petra.ac.id

Mauridhi Hery Purnomo
Dept. of Electrical Engineering, Dept. of Computer Engineering, Faculty of Intelligent Electrical and Information Technology (ELECTICS), Institut Teknologi Sepuluh Nopember and University Center of Excellence on Artificial Intelligence for Healthcare and Society (UCE AIHeS), Surabaya, Indonesia
hery@ee.its.ac.id

2021 International Electronics Symposium (IES) | 978-1-6654-4346-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IES53407.2021.9594024

***Abstract*—The most automated speech recognition (ASR) systems are extremely complicated, integrating many approaches and requiring a high variety of tuning parameters. Deep understanding and experience of each component are required to achieve optimal performance in ASR, confining the development of ASR systems to the experts. Hyperparameters are crucial for machine learning algorithms because they directly regulate the behavior of training algorithms and have a major impact on model performance. As a result, developing an effective hyperparameter optimization technique to optimize any given machine learning method would considerably increase machine learning efficiency. This work investigates the use of Random Forest and Bayesian to automatically optimize BiLSTM-CNN systems. We built the ASR based on the BiLSTM-CNN model and customized its hyperparameters value to heed our low-hardware specification during optimization. Furthermore, we gathered 1,000 clips of speech data from various movies, classifying them according to emotion and stress classes. In pursuit of contextual-level understanding in our ASR, we transcribed our speech data and used the bigram textual feature. Our Random Forest-optimized BiLSTM-CNN model ultimately reaches 84% of accuracy result and learning runtime in under 17 seconds.**

Keywords—Automatic Speech Recognition, Hyperparameter Optimization, BiLSTM-CNN, Random Forest, Bayesian Optimization

I. INTRODUCTION

Many researchers have widely studied Automatic Speech Recognition (ASR), successfully implemented in speech-related applications. Since ASR transforms human speech signals to sentence as text, several tasks processing human speech information adopt speech recognition results. The three basic foundations of a speech recognition model are Basis on Acoustic (BA), Basis on Pronunciation (BP), and Basis on Language (BL) [1]. BA learns how each syllable is spoken and sequentially organized in speech production since human language comprises different syllables. BP stores information on how words are mapped to their actual pronunciation, influenced by the language's complicated phonological laws. Finally, BL stores the grammatical information and correct word sequence patterns that make a phrase natural. A dictionary must be used to create a BP since each word must be appropriately mapped to phoneme sequences. As a result, developing a vocabulary is critical for building a whole process. However, this process is complex and time-consuming because it must be updated anytime new words are discovered in a corpus and cover various pronunciations if each word is spoken in multiple ways. For example, bear (animal) has two ways to be pronounced: [bɛːa] or [beuh]. Trigram modeling is one of the most effective ways to construct a BL. However, there are certain disadvantages, such as a sparse representation of language and processing cost [2].

Machine learning entails predicting and classifying data and employing various machine learning models based on the dataset. Machine learning models are programmable so that their behavior can be modified for a specific problem. These models have numerous parameters, and determining the best set of parameters can be viewed as a search issue [3].

This paper is organized as follows: Section 1 introduces the importance of using hyperparameter optimization. Section 2 supports the ASR idea by previous work related to optimized deep learning models. Section 3 illustrates our experiment setup, from data collection, data pre-processing, features extraction, and optimization of BiLSTM-CNN using Random Forest. Section 4 provides results of accuracy, runtime, tuned hyperparameter, and comparison with other works.

Ultimately, we conclude our optimized BiLSTM-CNN in Section 5.

II. Related Work

A. Deep Learning Models

Convolutional Neural Network (CNN) is initially powerful enough to map the feature space's structural locality. As CNN applies the local frequency region pooling, it is able to handle a small shift in feature space and translational variance. CNN also works very well with image data, while CNN might be used for speech and text data [4]. The ASR system usually converts speech data into spectrogram data or cochleagram data using the Hilbert-Huang transform to bring the most of CNN. But Soltau et al. argue that the CNN system deteriorates with semi-clean data [5].

One solution is to combine CNN with Recurrent Neural Network (RNN). Generally, RNN often yields high recognition accuracy in noise-robust contextual data [6]. Unfortunately, RNN has a burdening runtime because of the Vanishing and Exploding Gradients (VnEG) phenomenon [7]. The accumulation of large derivatives in exploding gradients results in the model being volatile and ineffective in the successful learning stage. The drastic changes in the model's weights establish a precarious and chaotic network, which at extreme values causes the inundation. It often results in null ("Not a Number" data type or NaN) values for weights. The accumulation of small gradients, on the other hand, results in a model that is incapable of comprehending meaningful insights because of the weights and biases of the initial layers. These biases tend to ineffectively updates the weight from the input data's features. In the worst-case scenario, the gradient will yield zero value, causing the network to stop further training [7].

The memory unit in Long-Short Term Memory (LSTM) tends to tackle the aforementioned problem. The memory in LSTM is a special unit that arguably controls the flow of information and how weights are updated [8]. It has a natural sensitivity of static data, which can be solved by tinkered in the model to generate low latency between input and direct output. This solution is beneficial for acoustic modeling tasks. It is often offered by special architecture called the Bidirectional LSTM, which controls the input in both directions between LSTM layers. Ayutthaya and Pasupa suggested using the BiLSTM-CNN method for standard sentiment analysis and speech emotion systems [9]. Their experiment specifically applies the Thai language and successfully reaches an accuracy of 78.89%. However, Ayutthaya and Pasupa only use textual data.

We implement two BiLSTM layers and one CNN layer in this work. Our optimized model of BiLSTM-CNN takes input from both speech and text data by handling the translational variance (tackled by CNN) and the VnEG problem (tackled by BiLSTM) almost simultaneously. We involved about 200 speech features and bigram text features that connect two words in a sentence. Later on, we use Weighted Average Accuracy validation methods and runtime for both hyperparameter optimization and end-stage learning that uses the best fit hyperparameter model. To the best of our knowledge, runtime as validation is important in such recognition systems to reveal its ability to tackle large data and/or complex models [10].

B. Hyperparameter Optimization Concept

The iterative machine learning component is essential because models may adjust autonomously as they are exposed to fresh captured data. Because machine learning relies on solving mathematical models effectively, optimization is a necessary element of the process [11]. On the other hand, machine learning can give novel momenta and optimization suggestions by reducing the computing costs of optimization, one of the most significant artificial intelligence applications. Various methods for solving issues based on assumptions have been given and explored, each with its own set of assumptions.

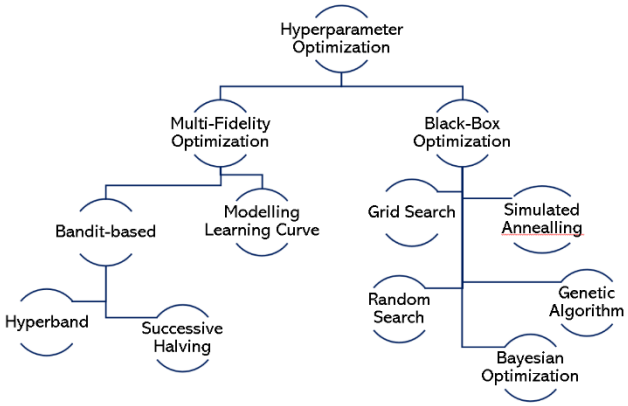

Fig. 1. A classification of hyperparameter optimization methods [12]

Hyperparameter optimization is frequently confused with automated machine learning because of its widespread use and ease of conceptualization [11], [13]. There are a few crucial factors while considering hyperparameters. The default settings of hyperparameters have long been known to be inefficient. Depending on the methodology, the hyperparameter tuning process usually improves accuracy by 3–5%. Parameter tweaking resulted in up to a 60% increase in accuracy in some cases and datasets [14]. There are several ways to finding the proper continuous hyperparameters, for example:

- A model's learning rate
- Hidden layers number's
- Batch size of iterations
- Number of iterations

The kind of operator, activation function, and method choice are all examples of categorical hyperparameters, also can be conditional, such as choosing the convolutional kernel size if a convolutional layer is utilized, or the kernel width if an SVM uses a Radial Basis Function (RBF) kernel. There are many different hyperparameter optimization approaches since there are many different types of hyperparameters. The following techniques are utilized for hyperparameter optimization: grid and random search, evolutionary-based, bandit-based, Bayesian optimization, and Gradient Descent-based techniques [15]. Manual, grid, and random search are the three most straightforward hyperparameter tuning approaches, as seen in Figure 1. Manual turning is dependent on intuition and guesswork based on previous experiences, as the name implies. Grid and random searches are slightly

different when choosing a set of hyperparameters for each combination (grid) or randomly cycle over them to keep the top-performing ones. However, as the search space grows more prominent, the optimization can rapidly become computationally unmanageable [15]. Furthermore, simple illustrations of grid search and random search are shown in Figure 2.

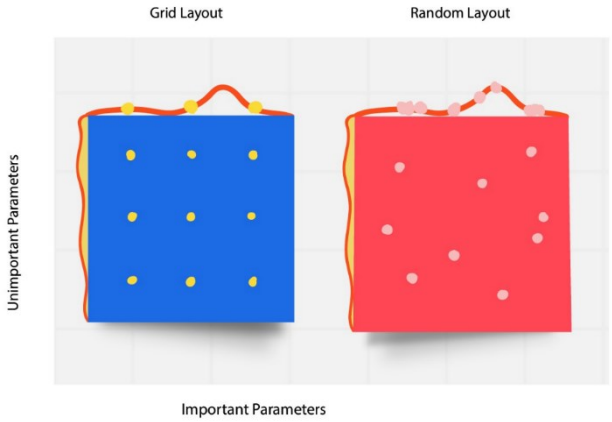

Fig. 2. Illustration of Point Grids and Random Point Sets

C. Hyperparameter Optimization with Random Forest

There is a popular hyperparameter search technique, namely the grid search. Sadly, it almost always hardly struggle to adapt to high dimensions [16]. Therefore, a substantial amount of newer studies have concentrated on superior methods, namely Bayesian Optimization (Bayes-Opt) and its derivative, namely the Random Forest-based Optimization (RF-Opt). The RF-Opt follows the sequential version of Bayes-Opt (Sequential-based Model of Bayesian Optimization or SMBO). The RF-Opt derivation is later called the Sequential Model-based Algorithm Configuration (SMAC) for general Deep Learning models.

Algorithm 1. RF-Opt Algorithm

Input: Target f^X; limit H; hyperparameter space Ψ; initial space $\langle \Psi_1, \dots, \Psi_t \rangle$

Result: Best hyperparameter configuration as $\widehat{\Psi}$

for $i + 1$ to h
do $y_i \leftarrow$ **evaluate** $f^X(\Psi_i)$
for $j \leftarrow$ to $h + 1$ to H
do $\mathcal{M} \leftarrow$ fit model on performance data $\langle \Psi_i, y_i \rangle_{i=1}^{j-1}$
select $\Psi_j \in arg\ max_{\Psi \in \Omega}\ \alpha(\Psi, \mathcal{M})$
$y_j \leftarrow$ **evaluate** $f^X(\Psi_j)$
end for
return $arg\ min_{\Psi_j \in \Psi_1, \dots, \Psi_T} y_j \xrightarrow{yields} \widehat{\Psi}$

The configuration is structured as follows: suppose we have X as the dataset. Also, suppose the $\Psi_1, \dots, \Psi_n$ denotes the hyperparameters of BiLSTM and CNN, and suppose $\Omega_1, \dots, \Omega_n$ denote the value of their numerical and categorical parameters. The RF-Opt in our work denotes all hyperparameters of numerical (learning rate, nodes number, batch size, dropout value, epoch variation) and categorical (activation function, optimizer, weight initialization technique, loss optimization, etc.) in Ω_i. Thus, space algorithm for proposed BiLSTM-CNN hyperparameters is defined as $\Omega = \Omega_1 \times \dots \times \Omega_n$. When trained with $\Psi \in \Omega$ on data X_{train}, we denote $Z(\Psi, X_{train}, X_{valid})$ on data X_{train} as the algorithm's validation. Using the built-in k-fold cross-validation, the hyperparameters optimization problem for dataset X is shown in Equation (1).

$$f^X(\Psi) = \frac{1}{k}\sum_{i=1}^{k} Z\left(\Psi, X_{train}^{(i)}, X_{valid}^{(i)}\right) \quad (1)$$

The SMAC-based RF-Opt is an efficient tool for global optimization of costly black box functions f. A complete RF-Opt stage is defined in Algorithm (1). RF-Opt starts by function inquiry f to the h values in an initial space and record $\langle \Psi_i, f(\Psi_1) \rangle_{i=1}^{t}$ as the $\langle input, output \rangle$ result pair. Then it fit a probabilistic-based model $\mathcal{M}$ to the previous recorded. Later on, $\mathcal{M}$ is used to select input for Ψ, which happened to be evaluating function value from input $\Psi \in \Omega$ through acquisition function of $\alpha(\Psi, \mathcal{M})$. Finally, it evaluates function in Ψ newest input.

III. Experiment Setup

We divided our experiment into approximately five scenarios, which are shown in Table I. The scenario's division is based on each speech (BA and BP) and textual data (BL). Four of them are for text data (binary classification), and one is for speech data (multiclass classification). We predict stress's contextual word-level information related to the speech data's primary function in recognizing speech-based emotion. The "non-" version for every class in each scenario is taken from other classes.

TABLE I. Experiment Scenarios

Details	*Scenario*				
	1	*2*	*3*	*4*	*5*
Data	Speech	Text	Text	Text	Text
Data Class for Training Stage	Anger, Fear, Sad, Happy	Anger, Non-Anger	Fear, Non-Fear	Sad, Non-Sad	Happy, Non-Happy
Data Class for Testing Stage		Stress, Non-Stress			

Our speech emotion-stress recognition system receives speech input, and later wit.ai engine converted it into text data. The transcribed text data is intent-categorized as either Stress or Non-stress, based on context. Both speech data and text data undergo their own data pre-processing steps. Afterward, both data undergo the feature extraction step. We also optimized most of the influential BiLSTM-CNN hyperparameters.

A. Data Pre-processing

For speech data pre-processing, we used the sample rate generalization. We gathered about 1,000 clips of speech data from various movies. We also generalized all sample rates into 44,100 Hz. The silence segment trimming cuts the silence segment in the speech data so that there are no empty data segments when the speech features are extracted. Because whole speech data is taken from movies with a duration ranging from 1 to 2 hours, we have brought down every recording to be trimmed around 5 seconds. These clips are also used for text data creation in wit.ai, classified as Stress or Non-stress. Meanwhile, the duration trimming within the limits of the speech-to-text processing capabilities of the wit.ai engine is a maximum of 20 seconds.

At the same time, we used three different data pre-processing for textual features: punctuation removal,

whitespace removal, and number removal. It removes unnecessary punctuation symbol marks (!, @, #, $, %, ', ", etc.), removes unnecessary space in-between words or characters, and removes numbers that may have been automatically transcribed using wit.ai engine.

Furthermore, our speech data processed by wit.ai resulted in all lowercase utterances. The wit.ai engine has no recognition for specific subjects that should begin with the first character's upper case. Therefore, we eliminate the need for the lowercase convert function of standard data pre-processing. After serious consideration, we also do not use stopwords removal since many of our data included in stopwords are too influential to be removed.

B. *Feature Extraction*

We used the notably famous and robust Harmonic-to-Noise Ratio, Fundamental Frequency, Zero Crossing Rate, Root Mean Square Error, Mel-Frequency Cepstral Coefficient, and Linear Predictive Coding features as its derivations. We used Jitter and Shimmer that represent perturbation which often happens in jittery speech. Spectral features are also used to represent time-series or temporal speech. The Intensity and Loudness features are also vital in determining stressed intonation.

TABLE II. EXTRACTED SPEECH FEATURES

Feature Name	***Total Features w/ Derivation(s)***
Fundamental Frequency	15
Harmonic-to-Noise Ratio	15
Intensity	15
Jitter	30
Linear Predictive Coding	24
Loudness	15
Mel-Frequency Cepstral Coefficient	30
Root Mean Square Error	6
Shimmer	15
Spectral Centroid	6
Spectral Flux	6
Spectral Rolloff	6
Spectral Variability	6
Zero-Crossing Rate	15
Total	204

Table II shows the complete 204 speech features extracted in this work that are related to emotion. Meanwhile, we used the bigram feature for text data to grasp two-word pair worth of contextual stress recognition.

C. *Customizing Hyperparameter Value*

Table III illustrates the value for each hyperparameter, either in numerical or categorical. The customization of hyperparameter value relies on how we do not take every value to the respective hyperparameter, therefore reducing the probability and is beneficial for low-budget hardware [17]. The best fit model chosen by RF-Opt within the said value will automatically apply in BiLSTM-CNN in each experiment scenario. Any possible patterns of hyperparameters value combination are limited to 100 iterations (with no early stopping employed) to not further burden the hardware.

TABLE III. HYPERPARAMETERS VALUE TO BE OPTIMIZED

Hyperparameter	***Value***
Activation Function	sigmoid, tanh, relu, leaky_relu, selu, linear, softmax
Batch Size	2 until 64
CNN Filter	2 until 10
Data Test Size	0.4, 0.3, 0.2, 0.1
Data Train Size	0.6, 0.7, 0.8, 0.9
Dense Nodes Number	5 until 512
Dropout	2e-6 until 0.5
Epoch	30, 60, 70
L2 Kernel Regularization	0.01, 0.001, 0.0001, 0.000001
Learning Rate	1e-6 until 1e-1
Loss Function	binary_crossentropy, categorical_crossentropy, hinge, mae
LSTM Unit	2, 4, 8, 16, 32, 64
Optimizer Function	Adam, RMSprop, Ftrl, SGD, AdaDelta, Adagrad
Weight Initialization	random_uniform, glorot_uniform, lecun_uniform, truncated_normal

IV. RESULT AND DISCUSSION

A. *Result*

As a result of optimization by RF-Opt, Figure 3 and Figure 4 shows every RF-Opt and Bayes-Opt runtime result for each tuning runtime and learning runtime in seconds.

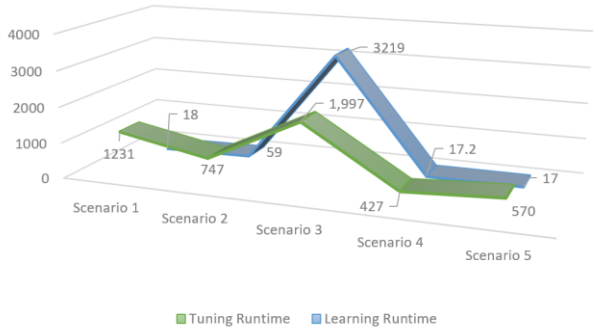

Fig. 3. BiLSTM-CNN Runtimes optimized by RF-Opt (in seconds)

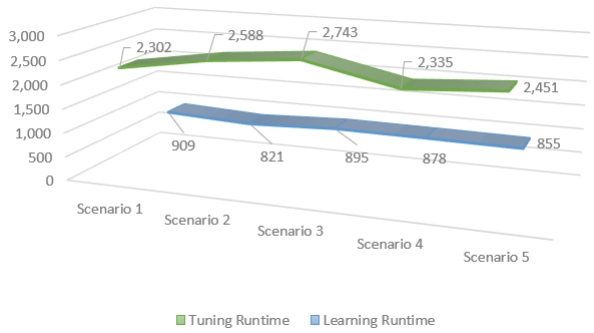

Fig. 4. BiLSTM-CNN Runtimes optimized by Bayes-Opt (in seconds)

TABLE IV. BEST HYPERPARAMETER OF BILSTM-CNN WITH RF-OPT

Hyperparameter	*Scenario 1*	*Scenario 2*	*Scenario 3*	*Scenario 4*	*Scenario 5*
Activation Function	softmax	tanh	sigmoid	relu	selu
Batch Size	19	10	14	21	16
CNN Filter	2	7	7	8	3
Data Train/Test	0.1/0.9	-	-	-	-
Dense Nodes Number	12	505	217	494	400
Dropout	0.002580695379659	0.09813176944163	0.049899	0.005835585171253	0.2217270034368
Epoch	60	60	60	60	30
Learning Rate	1.6e-03	0.01	0.000001	1e-06	1e-06
Loss Function	categorical_crossentropy	binary_crossentropy			
LSTM Unit	12	32	32	32	32
Optimizer Function	Adam			Ftrl	RMSprop
Weight Initialization	lecun_uniform	random_uniform	glorot_uniform	lecun_uniform	truncated_normal

TABLE V. BEST HYPERPARAMETER OF BILSTM-CNN WITH BAYES-OPT

Hyperparameter	*Scenario 1*	*Scenario 2*	*Scenario 3*	*Scenario 4*	*Scenario 5*
Activation Function	selu	relu	tanh	tanh	sigmoid
Batch Size	26	3	11	23	12
CNN Filter	3	10	8	6	6
Data Train/Test	0.1/0.9	-	-	-	-
Dense Nodes Number	305	200	214	267	288
Dropout	0.0015794629720744	2.069459761472289e-05	0.099124155	0.63231827967	0.701651690311
Epoch	30	50	50	30	60
Learning Rate	1.1e-03	1.6e-04	1.4e-05	1.2e-07	1.7e-05
Loss Function	categorical_crossentropy		hinge	binary_crossentropy	mae
LSTM Unit	16	32	12	12	16
Optimizer Function	RMSprop	Adam	RMSprop		Adam
Weight Initialization	truncated_normal	orthogonal	glorot_uniform	lecun_uniform	random_uniform

Table IV and Table V illustrate the hyperparameter tuning result of RF-Opt and Bayes-Opt, respectively, for each scenario.

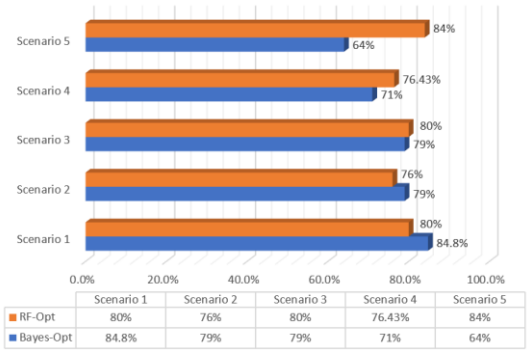

Fig. 5. Optimized BiLSTM-CNN Accuracy Result

Figure 5 shows a fairly consistent accuracy result for each scenario, between the respective optimization method of RF-Opt and Bayes-Opt. As presented in Table VI, our proposed work of optimized BiLSTM-CNN by RF-Opt and Bayes-Opt reached the best accuracy from previous work.

TABLE VI. RESULT COMPARISON WITH PREVIOUS WORKS

Authors	*Data*	*Testing Model*	*Best Accuracy*
[9] Ayutthaya et al. (2018)	Text	BiLSTM-CNN	78.89%
[18] Buddhika et al. (2018)	Text	NN, SVM, and DT	74%
[19] Zhang et al. (2018)	Text (Named Entity)	BiLSTM-CNN optimized by Gaussian	83.2%
[20] Qaffas (2019)	Text	Jaccard and Cosine	70%
[21] Duwairi et al. (2021)	Text	BiLSTM-CNN	73%
Proposed Model (2021)	**Text and Speech**	**BiLSTM-CNN optimized by RF-Opt**	**84%**
	Text and Speech	**BiLSTM-CNN optimized by Bayes-Opt**	**84.8%**

B. Discussion

The accuracy result, runtimes, and best hyperparameter are shown varied between RF-Opt and Bayes-Opt. In RF-Opt, the best activation function, batch size, CNN filter, dense number, dropout, learning rate, and weight initialization for every scenario is not the same, varied with no regard to the data. However, RF-Opt did pick binary_crossentropy as the

best loss function hyperparameter in most scenarios (4 out of 5), including at Scenario 5, which results in the best accuracy from RF-Opt (84%). The learning runtime for RF-Opt also seems to slow in Scenario 3, differs at 3,202 seconds by best learning runtime from Scenario 5, which is only 17 seconds.

Most numerical and categorical hyperparameter results by Bayes-Opt also seem to vary with no regard to speech or text data. Bayes-Opt showed consistent learning and hyperparameter tuning runtimes but is generally slower than RF-Opt, with the best runtime is at the learning runtime of Scenario 2 in 821 seconds. The best accuracy by Bayes-Opt is with Scenario 1 with 84.8% and decreases in each scenario.

V. Conclusion

The whole framework of our proposed speech emotion and stress recognition with BiLSTM-CNN is automatic since the data collection, feature extraction, hyperparameter optimization using RF-Opt, BiLSTM-CNN construction, and eventually end-result from accuracy and runtimes. Our automatic speech emotion and stress recognition took in two affective inputs of speech (emotion as BA & BP) and intent-based text (stress as BL).

RF-Opt optimized the proposed BiLSTM-CNN to achieve excellent accuracy results and fast learning runtime (17 seconds). Based on the scenario, RF-Opt did improve accuracy results up to 7,57%, from Scenario 4 with 76,43% to Scenario 5 with 84% accuracy. The runtime also improves from scenario to scenario, although both hyperparameter optimization and learning runtime are slowed at Scenario 3. For our future work, we planned on creating a data-centered custom hyperparameter from feature extraction with optimization from different SMAC-based methods.

Acknowledgment

We thank the Kementerian Riset dan Teknologi (Indonesian Ministry of Research and Technology) and the Badan Riset dan Inovasi Nasional (Indonesian National Research and Innovation Agency) for providing the research grant with contract number 848/PKS/ITS/2021.

References

[1] B. Shillingford *et al.*, "Large-Scale Visual Speech Recognition," *arXiv Prepr. arXiv1807.05162*, 2018.

[2] J. Fritsch, S. Wankerl, and E. Nöth, "Automatic Diagnosis of Alzheimer's Disease using Neural Network Language Models," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5841–5845.

[3] S. Paul, "Hyperparameter Optimization in Machine Learning Models," *Python Tutorials*, 2018. https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models? (accessed Jun. 01, 2021).

[4] H. Xie, S. Fang, Z.-J. Zha, Y. Yang, Y. Li, and Y. Zhang, "Convolutional Attention Networks for Scene Text Recognition," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 15, no. 1s, pp. 1–17, Feb. 2019, doi: 10.1145/3231737.

[5] H. Soltau, H.-K. Kuo, L. Mangu, G. Saon, and T. Beran, "Neural Network Acoustic Models for the DARPA RATS Program," in *INTERSPEECH*, 2013, pp. 3092–3096.

[6] R. Sharma, R. K. Bhukya, and S. R. M. Prasanna, "Analysis of the Hilbert Spectrum for Text-Dependent Speaker Verification," *Speech Commun.*, vol. 96, no. December, pp. 207–224, 2018, doi: 10.1016/j.specom.2017.12.001.

[7] A. H. Ribeiro, K. Tiels, L. A. Aguirre, and T. Schön, "Beyond Exploding and Vanishing Gradients: Analysing RNN Training using Attractors and Smoothness," in *International Conference on Artificial Intelligence and Statistics*, 2020, pp. 2370–2380.

[8] J. Zhao *et al.*, "Do RNN and LSTM have Long Memory?," in *International Conference on Machine Learning*, 2020, pp. 11365–11375.

[9] T. S. N. Ayutthaya and K. Pasupa, "Thai Sentiment Analysis via Bidirectional LSTM-CNN Model with Embedding Vectors and Sentic Features," in *2018 International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*, Nov. 2018, pp. 1–6, doi: 10.1109/iSAI-NLP.2018.8692836.

[10] A. Sheth, H. Y. Yip, A. Iyengar, and P. Tepper, "Cognitive Services and Intelligent Chatbots: Current Perspectives and Special Issue Introduction," *IEEE Internet Comput.*, vol. 23, no. 2, pp. 6–12, Mar. 2019, doi: 10.1109/MIC.2018.2889231.

[11] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[12] R. Elshawi, M. Maher, and S. Sakr, "Automated Machine Learning: State-of-the-Art and Open Challenges," *arXiv Prepr. arXiv1906.02287*, 2019.

[13] M. Feurer and F. Hutter, "Hyperparameter Optimization," in *Automated Machine Learning*, Springer, Cham, 2019, pp. 3–33.

[14] R. S. Olson, W. La Cava, Z. Mustahsan, A. Varik, and J. H. Moore, "Data-driven Advice for Applying Machine Learning to Bioinformatics Problems," in *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2018: Proceedings of the Pacific Symposium*, 2018, pp. 192–203.

[15] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random search, Genetic Algorithm: A Big Comparison for NAS," *arXiv Prepr. arXiv1912.06059*, 2019.

[16] T. Yu and H. Zhu, "Hyper-parameter Optimization: A Review of Algorithms and Applications," *arXiv Prepr. arXiv2003.05689*, 2020.

[17] P. T. Sivaprasad, F. Mai, T. Vogels, M. Jaggi, and F. Fleuret, "Optimizer Benchmarking Needs to Account for Hyperparameter Tuning," in *International Conference on Machine Learning*, 2020, pp. 9036–9045.

[18] D. Buddhika, R. Liyadipita, S. Nadeeshan, H. Witharana, S. Javasena, and U. Thayasivam, "Domain Specific Intent Classification of Sinhala Speech Data," in *2018 International Conference on Asian Language Processing (IALP)*, 2018, pp. 197–202.

[19] L. Zhang and F. Xiang, "Relation Classification via BiLSTM-CNN," in *Data Mining and Big Data. DMBD 2018. Lecture Notes in Computer Science, vol 10943*, 2018, pp. 373–382.

[20] A. A. Qaffas, "Improvement of Chatbots Semantics using wit.ai and Word Sequence Kernel: Education Chatbot as a Case Study," *Int. J. Mod. Educ. Comput. Sci.*, vol. 11, no. 3, p. 16, 2019.

[21] R. Duwairi, A. Hayajneh, and M. Quwaider, "A Deep Learning Framework for Automatic Detection of Hate Speech Embedded in Arabic Tweets," *Arab. J. Sci. Eng.*, vol. 46, no. 4, pp. 4001–4014, Apr. 2021, doi: 10.1007/s13369-021-05383-3.