Performance Evaluation of KNIME Low Code Platform in Deep Learning Study and Optimal Hyperparameter Tuning

Pornpawee Thongkhome*, Takuro Yonezawa**, Nobuo Kawaguchi**
*Automotive Engineering - G30 International Program in Electrical, Electronic and Information Engineering
School of Engineering, Nagoya University, Aichi, Japan
**Graduate School of Engineering, Nagoya University, Aichi, Japan
pornpraweeth@gmail.com , takuro@nagoya-u.jp , kawaguti@nagoya-u.jp

Abstract— A low-code platform is a software development environment that allows for the creation of applications through graphical user interfaces and configuration instead of traditional hand-coded computer programming. In this study, the application to classify a dataset of traffic sign images using the KNIME low-code deep learning development platform will be discussed to represents this software performance especial in term of model optimization processes. By creates the workflow to perform image preprocessing, create the CNN layer under KERAS sequential API and finding the best set of key hyperparameters among traditional KNIME build-in optimization algorithm including Brute force, Hill climbing, random search, Bayesian Optimization and black-box optimizer Optuna optimization algorithm under 3 types CNN architecture as simple CNN, Resnet-50 and VGG16 to classify traffic sign images. The result demonstrates that both grid search and random search optimization can be effective, while both Optuna and Bayesian optimization stands out as a powerful method due to its ability to efficiently explore the hyperparameter space and achieve superior results to meet 99% accuracy under simple CNN environment, but Optuna is significantly improve optimization times than Bayesian about 7 - 8 times. The KNIME low-code platform provides a user-friendly environment for developing and fine-tuning models to contribute the ongoing progress in machine learning and deep learning research development.

Keywords – KNIME low code platform , Convolution neural network(CNN), Keras and TensorFlow API, Optuna, Hyperparameter Optimization.

I. INTRODUCTION

Deep learning optimization has recently seen significant advancements, with various methods emerging to enhance model performance. One prominent approach involves the optimization of training algorithms by adjusting hyperparameters like batch size, epochs, etc. The traditional tools such as Optuna, Bayesian Optimization, etc are commonly employed for this purpose. However, improving model accuracy through optimization alone is not always guaranteed, as the underlying issue may reside within the algorithm's architecture itself. Consequently, it is essential to adapt and optimize the algorithm to suit each specific dataset. Moreover, making optimization tools more accessible and user-friendly can significantly reduce the barrier of coding skills, thereby democratizing the deep learning field and fostering broader participation and innovation. In this context, the use of low-code platforms, which researcher explores the potential of the KNIME (Konstanz Information Miner) low-code platform, a widely used open-source tool for data analysis and data science with extensive extensions and integrations. We aim to evaluate the usability and effectiveness of KNIME in the domain of deep learning optimization.

To validate this approach, we selected traffic sign image datasets as the case study for image processing tasks. Convolutional neural networks (CNNs) were employed to enable accurate classification. Our research encompasses two key aspects of deep learning optimization: first, the design of a novel algorithm tailored specifically for the selected datasets; and second, the comparison with existing architectures, such as ResNet-50 and VGG-16, to assess the benefits of structural optimization, various traditional optimization technique with utilizing ADADELTA and SGD optimizers. A notable contribution of this work is the implementation of Optuna within KNIME, introducing a new approach to algorithm optimization within a low-code environment. The evaluation results indicate that the combination of Optuna and Bayesian Optimization applied to the customized architecture achieved a remarkable 99% accuracy. Despite yielding similar results, Optuna demonstrated superior time efficiency and performance compared to Bayesian methods, largely due to its seamless integration as a KNIME node. Many studies that deployed traffic sign image recognition have been done on the application of CNN[1],[2] and [3]. All of them have varied various CNN structures, to perform training algorithm hyperparameters tuning with traditional optimization techniques such as Bayesian optimization, Optuna optimization, etc. The results achieve the average accuracy scores > 99%.

In summary, we present a novel integration of Optuna within KNIME, offering an alternative method for performance evaluation and customize algorithm design for structural optimization. We demonstrate and reveal that KNIME enables effective structural optimization of algorithms, addressing a gap in existing low-code tools and provides a user-friendly environment equipped with a variety of tools for training algorithm optimization, promoting accessibility in the deep learning field.

II. Research Methodology

A. *Traffic sign dataset and CNN Implementation*

In this study, the two categories of traffic sign dataset selection for speed limit 100km/h and speed limit 120km/h (2730 images for 100km/h and 2670 images for 120km/h) from totally 42 categories are shared from KAGGLE. The original image is a color JPEG - RGB image with dimensions of 32 x 32 pixels. Researcher use KNIME for image data preprocessing, feature engineering, and data visualization, and then pass the prepared data to a deep learning framework that run under KNIME node for model training and evaluation. With a abundant of useful deep learning algorithm nodes, model optimization tools, and low-coded properties, it assists developers in significantly reducing development times. When comparing KNIME with other low-code platform software such as RapidMiner, Alteryx, SAS, etc [4]. KNIME has pricing flexibility, a standalone user could use it for free, and there are no limits to the software's performance.

The researcher create KNIME workflow and adjust KNIME node properties to easier change the relevant parameters and finding the best set of key hyperparameters that associated with training process improvement as learning rate, batch size, number of epochs and 2 optimizer; ADADELTA and Stochastic Gradient Descent (SGD) to compare performance of 3 types CNN as simple CNN, Resnet-50 and VGG16 to classify traffic sign images.

The first stage of CNN implementation with KNIME workflow involves data preprocessing to read the image file and decode each image file name for classification group labeling (Supervised Learning) and data partitioning, which is used to divide image data into two groups. The first image group separation accounts for approximately 70% of the total image for model training by linear sampling while the remaining image is used for model testing.

The second stage is CNN deep learning model training/testing process as example in Fig.1.

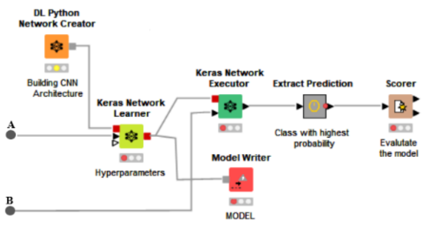


Fig. 1. Model training/testing KNIME workflow

The DL Python Network Creator node is used to create convolution base layer and dense layer by the KERAS Sequential API. The KERAS Network Learner node is used to train the created CNN from DL Python Network Creator node. The CNN layer architecture that use in this study are simple CNN , the standard Resnet-50 and the standard VGG16 that are often used in computer vision or image analysis applications such as image classification, object detection, face recognition, etc. In the last 2 nodes of this stage, we use DL Network Executor to test the acquired model by using the second group of partitioned image that was partitioned in the first stage and a model writer node is used to record the training model to keep in a local drive for future implementation.

The last stage is about the model evaluation. The KNIME scorer node is used to represent the confusion matrix and calculate accuracy of prediction output. It is a deep learning evaluation metric that assesses the predictive skill of a model by an overall performance due to the accuracy computes how many times a model made a correct prediction across the entire dataset, which remains valid if the dataset is class-balanced (refers to 2730 images for 100km/h and 2670 images for 120km/h in Section A).

B. *CNN hyperparameter tuning with KNIME node*

Hyperparameters are parameters whose values are set before the training process begins. Some key hyperparameters directly associated with CNN architecture such as number of convolutional layers, number of filters/kernels, filter/kernel Size, pooling type and size, activation function, etc. In additional, some key hyperparameters associated with training process improvement such as learning rate, batch size, number of epochs, dropout rate, optimizer, Weight Initialization, etc. Hyperparameter optimization is represented in equations (1) as.

$$x^* = \underset{x \in X}{\arg\min} f(x) \tag{1}$$

f(x) represents an objective score to minimize such as F1 score evaluated on the validation set, the equation (1) is the set of hyperparameters that return the lowest value of the scoring and x can take on any value in the domain X. The Fig.2. as below show hyperparameters optimization workflow in this study.

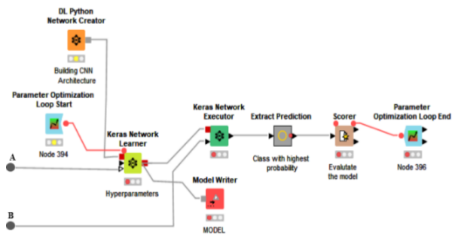

DL Python Network Creator
Building CNN Architecture
Parameter Optimization Loop Start
Node 394
Keras Network Learner
Hyperparameters
A
B
Keras Network Executor
Extract Prediction
Class with highest probability
Scorer
Evalutate the model
Parameter Optimization Loop End
Node 396
Model Writer
MODEL

Fig.2. KNIME parameter optimization workflow

Bayesian Optimization builds a surrogate model of the objective function and uses it to decide where to sample next. It is an efficient in terms of the number of evaluations needed, especially in high-dimensional search spaces. But it requires careful tuning of its own hyperparameters and may not perform well with noisy objective functions. So, the goal of Bayesian optimization is to find the global optimum of the objective function with as few evaluations as possible. A surrogate model (usually a Gaussian Process) is used to model the unknown objective function. This model provides a probabilistic estimate of the true objective function and is updated as more data points become available. Common Surrogate function used to model is the Gaussian Process (GP). Here in the Gaussian Process is described in detail.

$$f[x] \sim GP\left[m[x], k[x, x']\right] \tag{2}$$

Which,

$$E\left[f[x]\right] - m[x] \tag{3}$$

Given,

$$f = \left[f[x_1], f[x_2], \dots, f[x_t]\right] \tag{4}$$

at t points, the objective is to make prediction about the function value at a new point x*. This new function value $f^* = f[x^*]$ is jointly normally distributed with the observations f so that.

$$Pr\left(\begin{bmatrix} f \\ f^* \end{bmatrix}\right) = Norm\left[0, \begin{bmatrix} K[X,X] & K[X,x^*] \\ K[x^*,X] & K[x^*,x^*] \end{bmatrix}\right] \tag{5}$$

Where,

K[X ,X] is a t x t matrix and element (i,j) is given by k[x$_i$, x$_j$], K[X, X*] is a t x 1 vector where element i is given by k[x$_i$, x*] and so on. Since the values of the function in equation (5) are normal, the conditional distribution must be normal also, and the standard formula for the mean and variance of this conditional distribution becomes.

$$Pr(f^*|f) = Norm\left[\mu[x^*], \sigma^2[x^*]\right] \tag{6}$$

Where,

$$\mu[x^*] = K[x^*, X]\left[K[X,X]\right]^{-1} f \text{ and } \sigma^2[x^*] = K[x^*, x^*] - K[x^*, X]\left[K[X,X]\right]^{-1} K[X, x^*] \tag{7}$$

Using equation (6), the distribution of the function at any new point x* can be estimated. The best estimate of the function value is given by the mean μ[x] and the uncertainty is given by the variance σ^2[x].

An acquisition function is used to determine the next point to evaluate in optimization process. It balances the trade-off between exploration (sampling in regions where uncertainty is high) and exploitation (sampling in regions where the surrogate model predicts high values). Two typically used acquisition functions are the Probability of Improvement (PI) and the Expected Improvement (EI). PI measures the probability that the objective function value at the candidate point is better than the current best-known value. In each iteration, PI is maximized to determine the next point to evaluate. It encourages exploration by favoring points with a high probability of improvement over the current best-known value. The formulations of common acquisition functions, Upper Bound Confidence, Probability of Improvement and Expected Improvement are defined as equation (8,9,10) in respective below.

$$UCB[x^*] = \mu[x^*] + \beta^{1/2}\sigma[x^*] \tag{8}$$

$$PI[x^*] = \int_{f[x]}^{\infty} Norm_{f[x^*]}\left[\mu[x^*], \sigma[x^*]\right] df[x^*] \tag{9}$$

$$EI[x^*] = \int_{f[\hat{x}]}^{\infty} (f[x^*] - f[\hat{x}])\, PI \tag{10}$$

Optuna is define-by-run paradigm optimization tool that allows the user to dynamically construct the search space with efficient sampling and pruning algorithm that allows some user customization. It needs an objective function to decides where to sample in upcoming trials, and returns numerical values (the performance of the hyperparameters). Optuna uses Tree Parzen Estimator (TPE) by default to the efficient sampling or select the set of hyper-parameters to be tried next, based on the history of experiments. Optuna also provides another sampling strategy such as Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) to dynamically constructs the search space by updating the mean and variance of hyper-parameters. For k hyper-parameters, after N experiments, it use the best, say, 15% of the trials (best here is decided according to the metric of interest – accuracy). Optuna calculate the mean and covariance matrix of the joint distribution of these hyper-parameters. when estimating the covariance matrix, it use the mean of the previous generation (set of trials) instead of the mean estimated for this generation using previous trials as equation (11 - 15) as follow.

$$\mu_x^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} x_i \tag{11}$$

$$\mu_y^{(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} y_i \tag{12}$$

$$\sigma_x^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})^2 \tag{13}$$

$$\sigma_y^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (y_i - \mu_y^{(g)})^2 \tag{14}$$

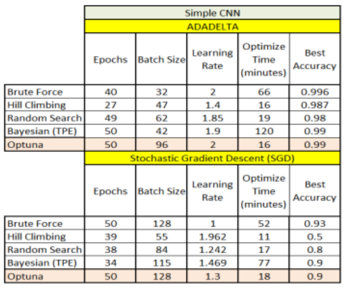
$$\sigma_{xy}^{2,(g+1)} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (x_i - \mu_x^{(g)})^2 (y_i - \mu_y^{(g)})^2 \tag{15}$$

Where x is parameter value, y is %loss, μ and σ are mean and variance of parameter in respective. Optuna saves optimization time with pruning that is a technique used in deep learning and searching algorithms to reduce the amount of decision trees, by removing some non critical branches of the decision tree that are redundant to classify instance. Optuna pruning will automatically stop unpromising trials at the previous stages of the training, call automated early-stopping. If an experiment seems unpromising based on some intermediate values of loss or other validation metric, the experiment is stopped. Optuna uses information from the previous experiment to make a decision. It asks what is the value of intermediate loss at this epoch, and what was the loss of the previous experiments at the same stage.

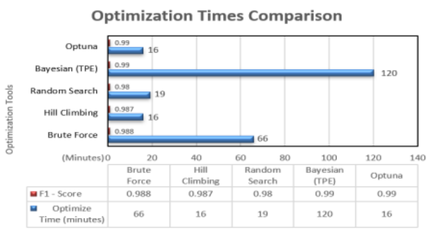
III. RESULT AND DISCUSSION

The result demonstrate that both Brute force (grid search) and random search can be effective, while Optuna and Bayesian optimization stands out as a powerful method due to its ability to efficiently explore the hyperparameter space and achieve superior results to meet 99% accuracy under CPU based and simple CNN environment, but Optuna is significantly improve optimization times than Bayesian about 7 - 8 times as shown in Table.I and Table.II. Additionally, the integration of the KNIME low-code platform in traffic sign recognition research enables streamlined development and optimization of deep learning models.

Table I. The accuracy and optimization times

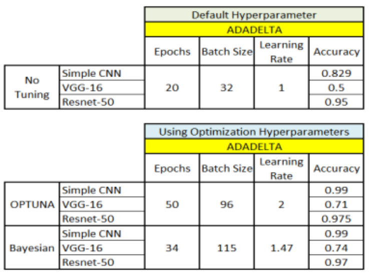


	Simple CNN				
	ADADELTA				
	Epochs	Batch Size	Learning Rate	Optimize Time (minutes)	Best Accuracy
Brute Force	40	32	2	66	0.996
Hill Climbing	27	47	1.4	16	0.987
Random Search	49	62	1.85	19	0.98
Bayesian (TPE)	50	42	1.9	120	0.99
Optuna	50	96	2	16	0.99
	Stochastic Gradient Descent (SGD)				
	Epochs	Batch Size	Learning Rate	Optimize Time (minutes)	Best Accuracy
Brute Force	50	128	1	52	0.93
Hill Climbing	39	55	1.962	11	0.5
Random Search	38	84	1.242	17	0.8
Bayesian (TPE)	34	115	1.469	77	0.9
Optuna	50	128	1.3	18	0.9



	Brute Force	Hill Climbing	Random Search	Bayesian (TPE)	Optuna
🟥 F1 - Score	0.988	0.987	0.98	0.99	0.99
🟦 Optimize Time (minutes)	66	16	19	120	16

Table II. The accuracy comparison that obtained from KNIME scorer node between each CNN layer architecture at before and after applying the important hyperparameters.

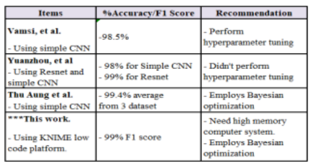


		Default Hyperparameter			
		ADADELTA			
		Epochs	Batch Size	Learning Rate	Accuracy
No Tuning	Simple CNN	20	32	1	0.829
	VGG-16				0.5
	Resnet-50				0.95

		Using Optimization Hyperparameters			
		ADADELTA			
		Epochs	Batch Size	Learning Rate	Accuracy
OPTUNA	Simple CNN	50	96	2	0.99
	VGG-16				0.71
	Resnet-50				0.975
Bayesian	Simple CNN	34	115	1.47	0.99
	VGG-16				0.74
	Resnet-50				0.97

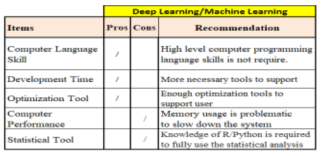
The comparison result with previous work is shown in Table III. The percent accuracy that get from this research when compared with previous works in term of accuracy is quite same. Due to, the different in the number of convolution layers and the number of dense layers in CNN have a direct impact on the system accuracy/F1 score for fairly comparison.

Table III. Previous work comparison result



Items	%Accuracy/F1 Score	Recommendation
Vamsi, et al. - Using simple CNN	-98.5%	- Perform hyperparameter tuning
Yuanzhou, et al - Using Resnet and simple CNN	- 98% for Simple CNN - 99% for Resnet	- Didn't perform hyperparameter tuning
Thu Aung et al. - Using simple CNN	- 99.4% average from 3 dataset	- Employs Bayesian optimization
*****This work.** - Using KNIME low code platform.	- 99% F1 score	- Need high memory computer system. - Employs Bayesian optimization

Table IV. Pros and Cons of KNIME



	Deep Learning/Machine Learning		
Items	**Pros**	**Cons**	**Recommendation**
Computer Language Skill	/		High level computer programming language skills is not require.
Development Time	/		More necessary tools to support
Optimization Tool	/		Enough optimization tools to support user
Computer Performance		/	Memory usage is problematic to slow down the system
Statistical Tool		/	Knowledge of R/Python is required to fully use the statistical analysis

This research represents the simple CNN that is the best design to recognize traffic sign due to researcher get the best key hyperparameters as learning rate, batch size, etc from this CNN type. Thus, we can't implement those value to the other CNN architecture. It should be optimize these hyperparameters by itself. In addition, preprocessing stage improvements such as the image filtering technique could be applied to increase the accuracy and KNIME already has many image preprocessing nodes to support this activity. The Table IV represents Pros and Cons of KNIME software for researcher who finding appropriate tool to significantly reduce development times in their data science research field. Bayesian optimization and Optuna emerges as a robust optimization method, while the KNIME low-code platform provides a user-friendly environment for developing and fine-tuning models.

ACKNOWLEDGEMENT

This research is partially supported by JSPS KAKENHI 23K24836.

REFFERENCES

[1] W.Yuanzhou and et al., "Research and Implementation of Traffic Sign Recognition Algorithm Model Based on Machine Learning", Journal of Software Engineering and Applications, Vol.16 No.6, June 2023.

[2] Aumg.Si Thu and et al., "Sequential Model-based Optimization Approach Deep Learning Model for Classification of Multi-class Traffic Sign Images", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 14, No. 7, pp.578 - 583, 2023.

[3] Takuya Akiba and et al.," Optuna: A Next-generation Hyperparameter Optimization Framework", The 25th International Conference on Knowledge Discovery and Data Mining(KDD'19), pp.2023,July 2019.

[4] Alteryx vs KNIME vs RapidMiner comparison Report,[Online]. https://www.peerspot.com/products/comparisons/alteryx_vs_knime_vs_rapidminer