Engineering and Applied Sciences
2022; 7(6): 93-99
http://www.sciencepublishinggroup.com/j/eas
doi: 10.11648/j.eas.20220706.13
ISSN: 2575-2022 (Print); ISSN: 2575-1468 (Online)

Vision Code Execution Time Prediction Based on Multi-level and Multi-scale CNN

Fule Ji[†, *], **Yanlong Xi**[†]

College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China

Email address:

jifule@hdu.edu.cn (Fule Ji), xyl@hdu.edu.cn (Yanlong Xi)

[*]Corresponding author

† Fule Ji and Yanlong Xi are co-first authors.

To cite this article:

Fule Ji, Yanlong Xi. Vision Code Execution Time Prediction Based on Multi-level and Multi-scale CNN. *Engineering and Applied Sciences*. Vol. 7, No. 6, 2022, pp. 93-99. doi: 10.11648/j.eas.20220706.13

Received: November 24, 2022; **Accepted**: December 8, 2022; **Published**: December 15, 2022

Abstract: Intelligent manufacturing relies heavily on industrial vision, and visual algorithms are rapidly being applied in the industry. However, industrial controllers are primarily used for logic control with deterministic execution cycles, and the uncertainty of vision code execution time strongly correlated with input affects their stability. To adjust the scanning cycle of the system in time to ensure system stability, an algorithm that can predict the time required for the vision code to process the target image is needed. In this paper, we analyze the weakness of traditional convolutional neural network models (CNN) and propose a multi-level and multi-scale CNN model (MLMS-CNN) for vision code execution time prediction. Instead of typical convolutional layers, we design an architecture to collect multi-scale features from the input feature maps. Moreover, a hierarchical structure is designed to reduce the loss of intermediate feature utilization by fusing features from different abstraction levels. We extract image features from images and runtime features from vision code blocks, then compare MLMS-CNN to six standard regression models, all of which are trained with the extracted features as input and the actual execution results of the visual code as output. The experimental results show that our model achieves better performance and stability.

Keywords: Deep Learning, Performance Prediction, Vision Code

1. Introduction

Vision algorithms are rapidly being used in intelligent manufacturing processes, such as defect detection of printed circuit boards [1], automatic flaw detection on the surface of steel parts [2], and intelligent sorting robots on logistics production lines [3]. Traditional industrial controllers, primarily performing logic control, have a predefined execution cycle. However, the introduction of vision algorithms not only increases the computational burden but also introduces unpredictability in the execution time. The maximum and bottom bounds of the execution time for some image recognition algorithms may vary significantly depending on the image content. The controller needs a method to predict the execution time of the target vision algorithm for the target image to adjust its scan period appropriately and ensure stability.

In terms of code execution time prediction, many research institutions at home and abroad have carried out extensive research and proposed a series of performance modeling methods. These methods can be categorized into three models: analytical models, replay models, and statistical models. Analytical models use formalized mathematical formulations to describe the program performance [4, 5, 34], which needs an in-depth understanding of the implementation of programs and the hardware characteristics of the platform and relies on the guidance of domain experts. The modeling procedure is time-consuming and laborious. Replay models automatically reconstruct a new program to reproduce the behavior of the original program and predict its performance by analyzing the historical program execution behavior records, avoiding reliance on manual analysis [6, 7, 20, 35]. The generated program fragment can restore the process of program execution and interaction with hardware, so this method has a good prediction effect on the performance of the original program. This method needs a lot of time and space to generate and process traces and can only

express one execution path of the original program. Statistical models predict program performance by establishing the mapping relationship between the program features and its performance indicators [8, 9, 22, 23]. With sufficient training data, program performance can be predicted with relative accuracy, but its prediction effect depends on the quality of the dataset as well as the consistency of the software and hardware environment in training and predicting procedure.

However, the existing performance prediction methods are mainly intended for general computing programs, and there is no approach for the vision algorithm with execution time strongly correlated with the content of the input image. This requires a feature extraction method for the input image to obtain image features related to algorithm execution time. In traditional image feature extraction, the Histogram of Oriented Gradients (HOG) [10] constructs features by calculating and counting the gradient direction histograms of local regions of the image, and is widely used in image recognition; Local Binary Patterns (LBP) [11] is used to describe the local texture features of images, which have the characteristics of rotation invariance and grayscale invariance; Haar features [12] can reflect the grayscale changes of images and are applied to face representation. However, traditional feature extraction methods are not suitable for all application scenarios. The emergence of deep learning has made it possible to extract image features for specific problems, and many mature models have emerged in this regard, such as VggNet [13], resnet [14], and DenseNet [15] and their variants are widely used in image feature extraction.

To this end, we propose a deep learning-based strategy to predicting performance of vision code blocks. The contributions of our study are as follows:

1) This paper proposes the multi-level multi-scale convolutional neural network (MLMS-CNN) for the runtime prediction of vision code blocks.
2) This paper proposes a dual feature extraction scheme based on image features and code runtime features, which obtains code runtime features through instrumentation technology [23] and extracts the image feature by Inceptionv3 migration training [33].
3) The result of independent replicate experiments shows that MLMS-CNN achieves excellent performance and stability.

The rest of this paper is organized as follows. Section 2 introduces the related work in performance modeling and convolutional neural networks. Section 3 discusses the two optimizations of CNN the framework of MLMS-CNN. Section 4 presents the experiment on model performance and stability. Section 5 concludes this paper.

2. Related Work

2.1. Performance Modeling

As mentioned before, the performance prediction methods can be classified into the following three categories: analytical modeling methods, replay-based modeling methods, and statistical modeling methods.

2.1.1. Analytical Modeling Method

The analytical modeling methods intend to formalize the execution procedure and execution platform. Altenbernd et al. [16] proposed a method modeling from the source code level and predicts the execution time on the target platform through the linear combination of the execution time of a single instruction of the program. Van den Steen et al. [17] modeled the performance of superscalar processor programs. Taking the microarchitecture-independent features of the program as input, only once analysis of the program, its performance on multiple target platforms can be well predicted. Jongerius et al. [18] model the performance of multi-core processor programs with vector instruction set extensions, including inter-core shared cache contention, memory bandwidth contention, and instruction level parallelism, with the better predictive performance achieved on the Intel Xeon and ARM Cortex-A15 platforms. Those approaches require a deep understanding of the underlying, and it is highly customized.

2.1.2. Replay-Based Modeling Method

The replay-based modeling method attempt to reproduce the behavior of the original program by analyzing the historical behavior record of the program execution. Zhang et al. [19] propose a performance prediction tool that predicts the performance of the original program by collecting and recording computation and communication events during program execution and generating a tiny program that can mimic the behavior of the original program. For IO-intensive programs, Hao et al. [6] use a more efficient trace merge algorithm and trace compression algorithm, and the generated benchmark program can accurately simulate the calculation, communication, and IO behavior of the original program. The behavioral consistency of the benchmark program with the original program determines the prediction accuracy of the original program's performance. Aaziz et al. [21] used the run-time data of the original program and the benchmark program to evaluate the similarity of the two programs by hierarchical clustering and realized the evaluation of the consistency of their behaviors. Those methods avoid the dependence on manual analysis and have a good prediction performance. But a large amount of time and space overhead limits its scope of application.

2.1.3. Statistical Modeling Method

Statistical modeling methods, in general, predict program performance more correctly than other methods. Pham et al. [22] model program execution time as a function that depends on cloud workflow input and cloud characteristics and is used to predict workflow task execution time for different input data in the cloud. Sun et al. [23] predict the execution time of a program under new input by modeling the correlation between its runtime characteristics and execution time. But this method only extracts the early behavior features of the program, and the performance of complex behavioral programs cannot be accurately predicted. In [24], the performance prediction task

of the complex program is divided into the prediction task of each atomic unit by using the modularization method. In practice, the difference between the training and prediction environments has a significant impact on the prediction effect.

2.2. Convolutional Neural Networks

Like traditional ANNs, CNNs are organized hierarchically, comprised of an input layer, several hidden layers, and an output layer. The only difference is convolution operations are employed in CNNs, with which CNNs get better performance in image feature extraction and suit more for image-based tasks. A convolution layer has several convolutional kernels, which are two dimensional matrices. In this layer, the convolution operations would be conducted on the input feature maps with convolutional kernels. The number of kernels determines the number of the output feature maps.

A 2D convolution operation for input X with kernel $k_{s\times t}^{m\times n}$ can be calculated as:

$$Y_{i,j} = (k_{s\times t}^{m\times n} * X)_{i,j} = \sum_{p,q} W_{p,q} X_{si+p,tj+q}, \quad (1)$$

Where Y is the output matrix, m,n is the height and width of the kernel, W is a matrix of shape $m \times n$, representing the weight of the kernel, s,t is the sliding step of the kernel in width and height. Usually, $s = t = 1$, which $k_{s\times t}^{m\times n}$ can be recorded as $k^{m\times n}$, L is a matrix of shape $M \times N$.

Since the width and height of each kernel is designed to be smaller than the input, i.e., $m < M, n < N$, each neuron in this layer is only connected to a small local region of the input. In other words, the receptive field of each neuron is small and equal to the kernel size.

The local connections ensure that the trained filter has the strongest response to local regions of the input, thereby exploiting the spatially local correlation of the input (for an input image, pixels are more correlated with nearby pixels than with distant pixels) [25]. At deeper layers of the network, convolutions generally extract more abstract patterns and higher-level information.

A spatial pooling operation would be applied after a set of convolutional layers to spatially down-sample the input feature maps to reduce the feature map size while preserving crucial information, which can detect more abstract features and cross-scale spatial context, thereby concentrating semantic information [26]. The application of pooling layer can reduce model parameters, thus reduce the amount of computation and the chance of overfitting. Like convolution, pooling is defined by kernel size, stride, and padding operations. A typical max pooling operation with a 2×2 kernel size and a stride of 2 can reduce the size of the input feature map by a factor of 4.

3. Framework

Traditional CNN follows a sequential structure, which has two characteristics: (1) Each convolution layer can only extract features with fixed scale. (2) There is only one computational path between input and output data. Therefore, only the features computed from the last convolutional layer can be used for regression, and feature maps in the middle are only used as the input of the next layer, these disadvantages lead to the loss of intermediate feature utilization. In the following two sections, we propose effective frameworks for these two problems, respectively.

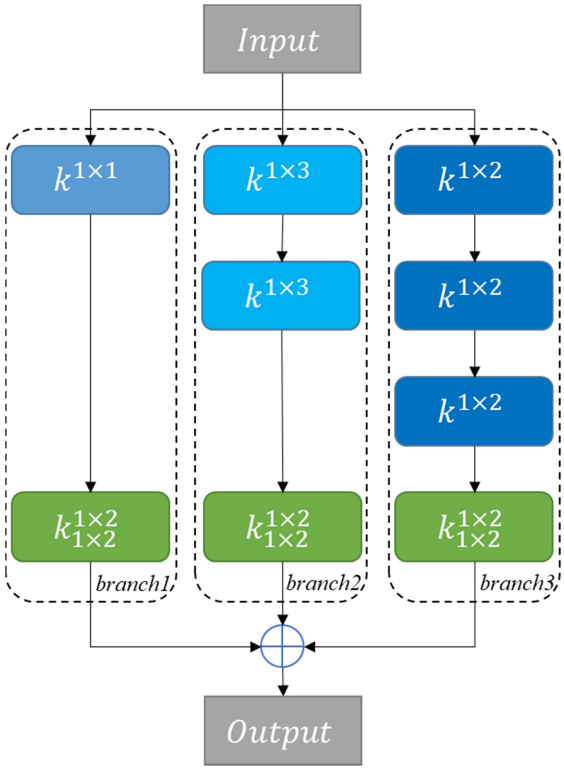

Figure 1. *Multi-Scale Block.*

3.1. Multi-scale Block

This section extends the convolution and pooling operations between layers and proposes a parallel structure, as shown in Figure 1, which we call Multi-Scale Block (MSB).

To obtain the features under different receptive fields, we create a basic kernel set $S = \{k_1, k_2 \ldots, k_m\}$, which contains m convolution kernels of various scales.

There are three parallel branches inside the block. Each branch contains different numbers and scales of convolution kernels from S, which can extract different levels of feature representations from receptive fields of various sizes. In addition, each branch has a down-sampling layer to reduce the dimension of the data. To avoid wasting feature information and ensure stable training, a special convolution kernel dsk is designed to replace pooling.

3.2. Multi-level Network Model

As shown in Figure 2, the Mulit-Level Network model (MLN) extracts features by introducing feature fusion. Supposing we have n layers $\{l_1, l_2, \ldots, l_n\}$, each layer l_i has one input in_i and one output out_i. We concat all outputs and fusion by full connection.

$$\mathrm{MLN}(in_1) = \mathrm{FC}\left(\mathrm{Concat}\begin{pmatrix} FC(\mathrm{out}_1), \mathrm{FC}(\mathrm{out}_2), \\ \ldots, \mathrm{FC}(\mathrm{out}_n) \end{pmatrix}\right) \quad (2)$$

$$in_{i+1} = out_i = l_i(in_i) \quad (3)$$

As in a traditional sequential network, the output of one layer is the input of the next layer, which means that the layers close to the in_1 are used to extract concrete features, while layers close to the out_n are used to extract abstract features. So, in a typical CNN, due to layered transmission, only abstract features can reach the final fully connected layer, which leads to the neglect of micro-scale information. Feature fusion mechanism can overcome this shortcoming by collecting information from various abstraction levels and directly weighting features at all resolutions in the fused fully-connected layers.

3.3. Multi-level and Multi-scale CNN Model

Based on the above two frameworks and the extracted code features to be introduced in detail in Session 4, we propose a multi-level and multi-scale convolutional neural network model (MLMS-CNN) to predict code runtime.

As shown in Figure 3, we choose $S = \{k^{1\times1}, k^{1\times2}, k^{1\times3}\}$, dsk is a $k^{1\times2}$ with 2 strides. Each feature generated by 5 MSBs will be flattened and connected to a fully connected layer of length 12. The output of these 5 fully connected layers will be further concatenated and injected into the fused fully connected layer with a length of 640 to obtain the final output.

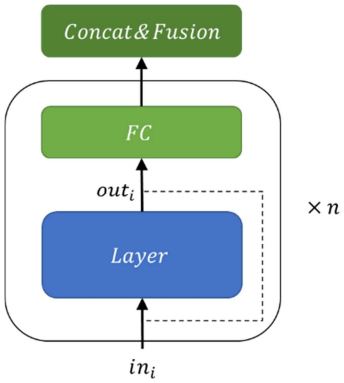

Figure 2. *Multi-Level Network model.*

4. Experiment

4.1. Setting

We perform a series of vision code block runtime prediction experiments on 500 pictures, which are split into train/test sets sequentially with a ratio of 8:2. The vision code block in experiments is Canny edge detection, which is a standard algorithm for edge detection developed by John F. Canny in 1986. Its internal structure is relatively complex, and it has many logical branches. So, we choose it as a representative vision code block for research. Before the experiment, we have obtained runtime features $RtF = \{rtf_1, rtf_2, \ldots, rtf_{64}\}$ of the Canny vision code block through instrumentation technology [23] and features of each image $IgF = \{igf_1, igf_2, \ldots, igf_{1000}\}$ by Inceptionv3 migration training [33].

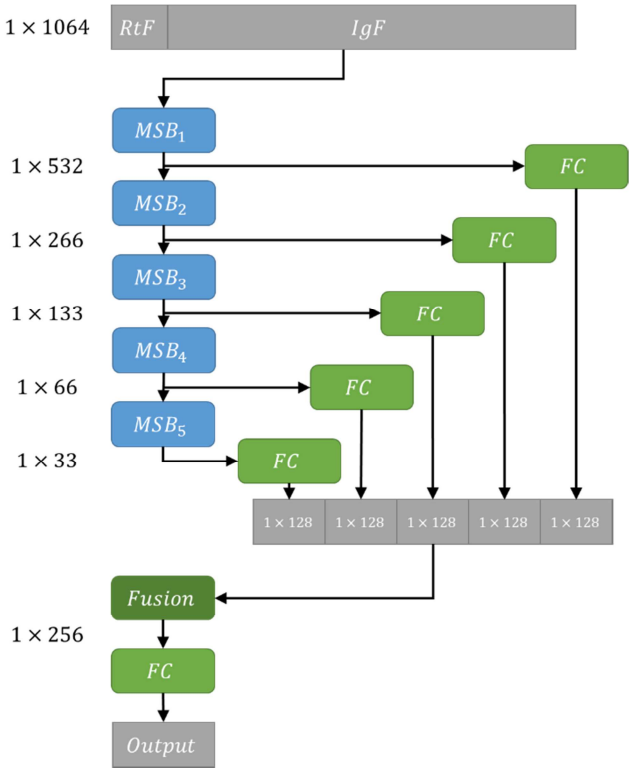

Figure 3. *MLMS-CNN architecture.*

4.2. Evaluation Metrics

R^2, Mean Squared Error (MSE), and Mean Absolute Error (MAE) are used as the evaluation metrics in the experiment. The definitions are shown in (4-6). The results of R^2 are normalized, which makes it easier to see the gap between models. MSE refers to the mean square error between the real and the predicted values, while MAE reflects the actual error value.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (4)$$

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \quad (5)$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}| \quad (6)$$

4.3. Result of MLMS-CNN

To explore the effect of the number of MSBs on the prediction performance, we conducted 5 experiments on each of the 6 models containing different numbers of multi-scale blocks and took the R^2 value for statistics. Particularly, the 0-MSB model is a simple CNN. As shown in Figure 4, when the number of MSBs increases, R^2 gradually increases, while the standard deviation of R^2 (performed as the length of the error bar) gradually decreases. When the number of MSBs is 5, the value of R^2 reaches 0.927. After that, the increase of the R^2 value slows down. This indicates that when the number of MSBs increases to 5, the training of the network parameters is

close to optimal, the ability of model regression tends to be saturated, and the benefit of continuing to increase the number of multi-scale modules begins to decline, which will increase the training time of the model. Therefore, we finally adopt 5 MSBs to construct the MLMS-CNN. This not only ensures the regression ability of the network but also reduces the parameters of the model, which is beneficial to shorten the training time of the model.

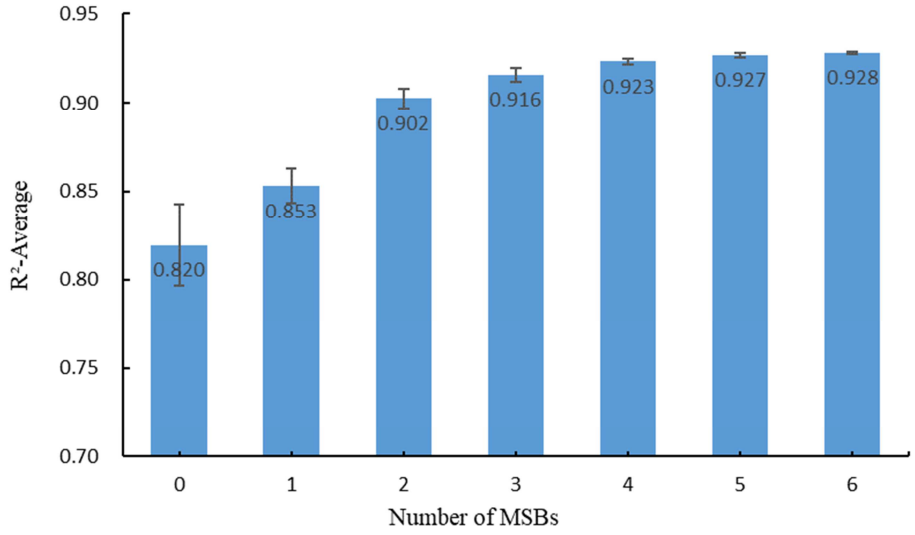

Figure 4. *Effect of the number of MSBs on the prediction performance.*

We also compare the model with MLN with the model without MLN, both have five MSBs, and both experiment five times. As shown in Figure 5, It can be intuitively known that multi-level feature fusion can facilitate the runtime prediction of vision code blocks. With the use of MLN, the value increased from 0.9274 to 0.9595 and the standard deviation decreased to 0.09%, indicating that the MLMS-CNN can not only enhance the utilization of features but also provide a more stable prediction of run times with a 50% increase in stability.

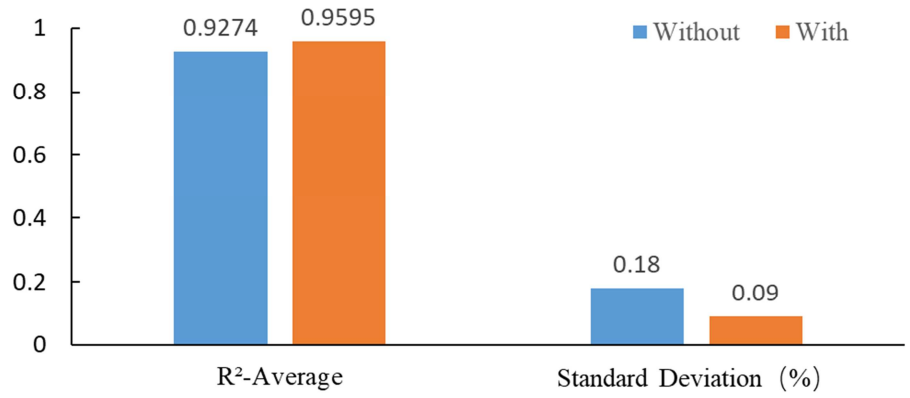

Figure 5. *Comparison of two models' performance.*

4.4. Comparison with Other Models

Table 1 shows the comparison results of MLMS-CNN with Partial Least-Squares Regression (PLSR) [27], Ridge regression [28], AdaBoost [29], Gboost [30], KNNR [31], SVR [32]. The R^2 of the SVR model is 0.107, the MSE mean square error is 1013.44, and the MAE average absolute error is 27.33. It is very unsatisfactory. The prediction effect of the KNNR model is better than that of SVR, which may be because the data set in this study is not linear. However, KNNR cannot effectively screen out interference features, so compared with other models, KNNR's performance is not excellent. Both PLST and Ridge are variants of the least squares method, and the least squares method is easily disturbed by outliers. PLST uses principal component analysis approach to reduce the dimensionality of the input data and reduce the influence of interference data, resulting in a slightly better performance than Ridge. Both AdaBoost and Gboost are evolutionary algorithms. They use model self-learning to regress data with large errors, gradually reduce the regression error, and gradually improve the regression performance of the model. It can be concluded from the table that AdaBoost and Gboost have certain advantages over the other 4 models. However, the proposed MLMS-CNN model outperforms other models in all three evaluations, which demonstrates the superiority of the architecture.

To eliminate the effect of randomness, we conducted 60

independent replicate experiments on the 6 existing models and the improved model proposed in this paper. The experimental results are shown in Figure 6. Because of the relatively poor prediction of SVR, it is not shown in the figure. The gap between the upper and lower quartiles of MLMS-CNN is small, and the lower edge is much higher than other models, indicating that MLMS-CNN's prediction effect is more concentrated and stable when predicting the runtime of vision code blocks.

Table 1. *Comparison with other models.*

	R^2	MSE	MAE
PLSR	0.919	91.753	7.537
Ridge	0.860	158.748	10.354
AdaBoost	0.930	78.937	6.776
Gboost	0.927	83.379	7.219
KNNR	0.752	281.634	14.463
SVR	0.107	1013.444	27.325
MLMS-CNN	0.960	45.974	5.428

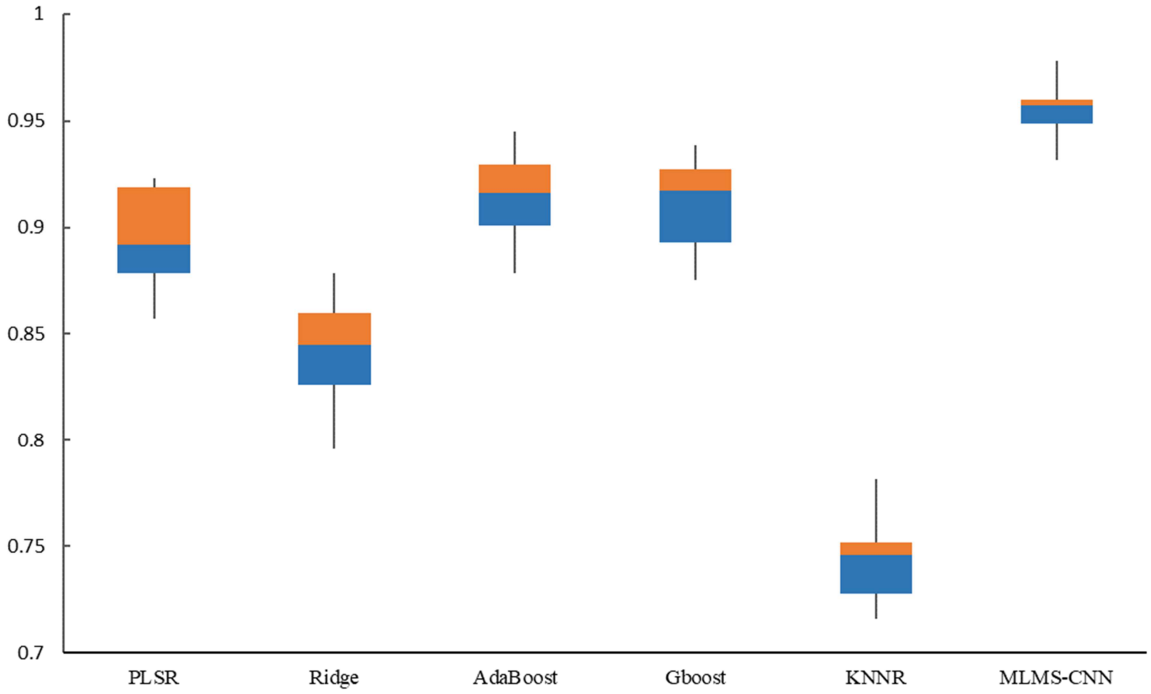

Figure 6. *Boxplot of Model Predictive Performance.*

5. Conclusion

This paper has presented a novel variant of CNN for vision code block runtime prediction. We design MSB to extract features from receptive fields of different sizes, and design MLN to fuse features of different abstraction levels. On this basis, we further propose MLMS-CNN for runtime prediction problem. The results of comparative experiments show that our MLMS-CNN obtains the best performance. In an additional 60 independent replicate experiments, MLMS-CNN still has a stable performance. In the further, we will further investigate how to predict the execution time of vision code blocks in different operating environments and further optimize our model.

References

[1] Lim D, Kim Y G, Park T H. SMD classification for automated optical inspection machine using convolution neural network [C]//2019 Third IEEE International Conference on Robotic Computing (IRC). IEEE, 2019: 395-398.

[2] Luo, Q., Sun, Y., Li, P., Simpson, O., Tian, L., & He, Y. (2018). Generalized completed local binary patterns for time-efficient steel surface defect classification. IEEE Transactions on Instrumentation and Measurement, 68 (3), 667-679.

[3] Abbood W T, Abdullah O I, Khalid E A. A real-time automated sorting of robotic vision system based on the interactive design approach [J]. International Journal on Interactive Design and Manufacturing (IJIDeM), 2020, 14 (1): 201-209.

[4] Barker K J, Pakin S, Kerbyson D J. A performance model of the krak hydrodynamics application [C]//2006 International Conference on Parallel Processing (ICPP'06). IEEE, 2006: 245-254.

[5] Kerbyson D J, Alme H J, Hoisie A, et al. Predictive performance and scalability modeling of a large-scale application [C]//Proceedings of the 2001 ACM/IEEE conference on Supercomputing. 2001: 37-37.

[6] Hao M, Zhang W, Zhang Y, et al. Automatic generation of benchmarks for I/O-intensive parallel applications [J]. Journal of Parallel and Distributed Computing, 2019, 124: 1-13.

[7] Sodhi S, Subhlok J, Xu Q. Performance prediction with skeletons [J]. Cluster Computing, 2008, 11 (2): 151-165.

[8] Huang L, Jia J, Yu B, et al. Predicting execution time of computer programs using sparse polynomial regression [J]. Advances in neural information processing systems, 2010, 23.

[9] Adams A, Ma K, Anderson L, et al. Learning to optimize halide with tree search and random programs [J]. ACM Transactions on Graphics (TOG), 2019, 38 (4): 1-12.

[10] Wang X, Han T X, Yan S. An HOG-LBP human detector with partial occlusion handling [C]//2009 IEEE 12th international conference on computer vision. IEEE, 2009: 32-39.

[11] Ojala T, Pietikainen M, Harwood D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions [C]//Proceedings of 12th international conference on pattern recognition. IEEE, 1994, 1: 582-585.

[12] Papageorgiou C P, Oren M, Poggio T. A general framework for object detection [C]//Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271). IEEE, 1998: 555-562.

[13] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition [J]. arXiv preprint arXiv: 1409.1556, 2014.

[14] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[15] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[16] Altenbernd P, Gustafsson J, Lisper B, et al. Early execution time-estimation through automatically generated timing models [J]. Real-Time Systems, 2016, 52 (6): 731-760.

[17] Van den Steen S, Eyerman S, De Pestel S, et al. Analytical processor performance and power modeling using micro-architecture independent characteristics [J]. IEEE Transactions on Computers, 2016, 65 (12): 3537-3551.

[18] Jongerius R, Anghel A, Dittmann G, et al. Analytic multi-core processor model for fast design-space exploration [J]. IEEE Transactions on Computers, 2017, 67 (6): 755-770.

[19] Zhang W, Cheng A M K, Subhlok J. Dwarfcode: a performance prediction tool for parallel applications [J]. IEEE Transactions on Computers, 2015, 65 (2): 495-507.

[20] Sieh V, Burlacu R, Hönig T, et al. Combining Automated Measurement-Based Cost Modeling With Static Worst-Case Execution-Time and Energy-Consumption Analyses [J]. IEEE Embedded Systems Letters, 2018, 11 (2): 38-41.

[21] Aaziz O, Cook J, Cook J, et al. A methodology for characterizing the correspondence between real and proxy applications [C]//2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2018: 190-200.

[22] Pham T P, Durillo J J, Fahringer T. Predicting workflow task execution time in the cloud using a two-stage machine learning approach [J]. IEEE Transactions on Cloud Computing, 2017, 8 (1): 256-268.

[23] Sun J, Sun G, Zhan S, et al. Automated performance modeling of HPC applications using machine learning [J]. IEEE Transactions on Computers, 2020, 69 (5): 749-763.

[24] Singh A, Purawat S, Rao A, et al. Modular performance prediction for scientific workflows using Machine Learning [J]. Future Generation Computer Systems, 2021, 114: 1-14.

[25] Ke, Q., Liu, J., Bennamoun, M., An, S., Sohel, F., & Boussaid, F. (2018). Computer Vision for Human-Machine Interaction. In L. Marco, & G. M. Farinella (Eds.), Computer Vision for Assistive Healthcare (pp. 127-145). Academic Press.

[26] Kattenborn T, Leitloff J, Schiefer F, et al. Review on Convolutional Neural Networks (CNN) in vegetation remote sensing [J]. ISPRS Journal of Photogrammetry and Remote Sensing, 2021, 173: 24-49.

[27] Hair J F, Sarstedt M, Ringle C M. Rethinking some of the rethinking of partial least squares [J]. European Journal of Marketing, 2019.

[28] Fan P, Deng R, Qiu J, et al. Well logging curve reconstruction based on kernel ridge regression [J]. Arabian Journal of Geosciences, 2021, 14 (16): 1-10.

[29] Shahraki A, Abbasi M, Haugen Ø. Boosting algorithms for network intrusion detection: A comparative evaluation of Real AdaBoost, Gentle AdaBoost and Modest AdaBoost [J]. Engineering Applications of Artificial Intelligence, 2020, 94: 103770.

[30] Yung L S, Yang C, Wan X, et al. GBOOST: a GPU-based tool for detecting gene–gene interactions in genome–wide case control studies [J]. Bioinformatics, 2011, 27 (9): 1309-1310.

[31] Cai L, Yu Y, Zhang S, et al. A sample-rebalanced outlier-rejected k-nearest neighbor regression model for short-term traffic flow forecasting [J]. IEEE access, 2020, 8: 22686-22696.

[32] Sharifzadeh M, Sikinioti-Lock A, Shah N. Machine-learning methods for integrated renewable power generation: A comparative study of artificial neural networks, support vector regression, and Gaussian Process Regression [J]. Renewable and Sustainable Energy Reviews, 2019, 108: 513-538.

[33] Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 2818-2826.

[34] Sundaram-Stukel D, Vernon M K. Predictive analysis of a wavefront application using LogGP [C]//Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming. 1999: 141-150.

[35] Lu G, Zhang W, He H, et al. Performance modeling for mpi applications with low overhead fine-grained profiling [J]. Future Generation Computer Systems, 2019, 90: 317-326.