An Enhanced Data Augmentation Approach to Support Multi-Class Code Readability Classification

Qing Mi*, Yiqun Hao, Maran Wu, Liwei Ou
Faculty of Information Technology, Beijing University of Technology, Beijing, China
Email: miqing@bjut.edu.cn, yiqun.hao@ucdconnect.ie, 3085179961@qq.com, liwei.ou@ucdconnect.ie

***Abstract*—*Context:* Code readability plays a critical role in software maintenance and evolvement, where a metric for classifying code readability levels is both applicable and desired. However, most prior research has treated code readability classification as a binary classification task due to the lack of labeled data. *Objective:* To support the training of multi-class code readability classification models, we propose an enhanced data augmentation approach. *Method:* The approach includes the use of domain-specific data transformation and GAN-based data augmentation. By virtue of this augmentation approach, we could generate sufficient readability data and well train a multi-class code readability model. *Result:* A series of experiments are conducted to evaluate our augmentation approach. The experimental results show that a state-of-the-art multi-class code readability classification accuracy of 68.0% is reached with a significant improvement of 6.3% compared to only using the original data. *Conclusion:* As an innovative work of proposing multi-class code readability classification and an enhanced code readability data augmentation approach, our method is proved to be effective.**

***Index Terms*—code readability classification; data augmentation; generative adversarial networks; program comprehension; software analysis**

I. Introduction

Being a critical factor affecting the maintainability and reusability of the software, source code readability, defined as the ease of understanding the source code [5], is growing crucial in modern software development with a higher demand for rapid deliveries. Specifically, recent research reveals that software developers spend nearly 59% of their development time on reading and understanding the source code before they start coding [23]. Thus, it is worthwhile to provide a tool that constantly monitors the readability of source code and urges developers to write code with high readability to shorten the time wasted [21].

Buse and Weimer opened up the code readability classification research in early 2008 using machine learning algorithms with critical factors mainly affecting code readability [5]. Later, many effective classification models are built including the use of deep learning techniques [13][14]. However, code readability classification is still far from practical use. Because most prior researches treat code readability classification as binary classification [6][13][14][19], that is identifying a piece of code as either readable or unreadable. It is not precise enough and too extreme to be applied for practical use. Therefore, this paper is a pioneering work to propose a deep learning-based multi-class code readability classification model.

However, the problem is that the size of the labeled readability dataset is merely acceptable for training a binary code readability classification model and far from being adequate for supporting a multi-class one. On top of that, using conventional methods to manually label data, such as conducting a large-scale survey to invite a number of programmers for data annotation, has always been too costly and inefficient [15]. Thus, we propose a data augmentation approach including a domain-specific data transformation method and a GAN-based (i.e. Generative Adversarial Network-based) data augmentation method to support the training of our deep learning-based multi-class code readability classification model.

The contributions of this paper are:

- To the best of our knowledge, we are the first to attempt multi-class code readability classification which classifies a code snippet as readable, neutral, and unreadable. This multi-class model is more suitable for practical use than existing ones.
- We propose a code readability data augmentation approach including domain-specific data transformation and GAN-based data augmentation. By utilizing the approach, labeled data could be generated from existing datasets along with a higher data diversity which could ultimately improve the performance of code readability classification models.
- We conduct a series of experiments to verify our proposed approach and gained a state-of-the-art multi-class code readability classification accuracy of 68.0%, f-measure of 67.3%.
- We publish all our data and source code online to benefit future researchers.[1]

This research is an extension of a short communication [15] which first proposed the use of data augmentation in code readability classification. There are several improvements made over the prior work:

- For the sake of enlarging the usable dataset to support multi-class classification, we refine the domain-specific data transformation method in a more systematic manner. Besides, WGAN rather than ACGAN is adopted in the GAN-based data augmentation method to get better stability.
- Apart from separately using two data augmentation methods, we propose a parallel augmentation method and a sequential augmentation method to combine the merits

*Corresponding author.
DOI reference number: 10.18293/SEKE2022-130

[1] https://github.com/swy0601/Code-Augmentation

of them and further explore the best data augmentation method in the field of code readability classification.
- A series of experiments are conducted with more evaluation metrics and classification models to comprehensively evaluate our data augmentation approach and make the results more cogent.

II. Related Work

In general, past code readability classification researches fall into two categories: machine learning-based and deep learning-based. Although they both produce a classifier, deep learning-based models could automatically extract readability features whereas machine learning-based models rely on handcrafted features pre-specified by researchers.

A. Machine Learning-Based Code Readability Classification

Buse and Weimer collected 100 code snippets and invited 120 online human annotators to label them based on a five-point Likert scale ranging from one (i.e., very unreadable) to five (i.e., very readable). By analyzing the dataset collected, a set of handcrafted code features that correlates with code readability (e.g., average number of identifiers) was produced. Then, those features are fed into machine learning algorithms to make code readability predictions. This preliminary code readability model successfully outperformed human judgments on average [5].

Subsequently, Posnett et al. [18] proposed a readability classification model based on two factors: code size and entropy, which outperforms Buse's model on the same dataset. However, Dorn argues that both of those models have a bad generalization ability because they only take surface features into consideration [6]. Therefore, Dorn incorporated geometric, pattern-based, and linguistic features and built another machine learning-based readability model.

Scalabrino et al. [19] enrich Dorn's model metrics by complementing textual aspects and achieved a better performance than all prior models. Apart from a better model, Scalabrino also manually labeled and added 200 pieces of readability data which constitute our ground-truth code readability dataset along with data from Buse and Dorn.

B. Deep Learning-Based Code Readability Classification

While machine learning-based models have gained relatively high accuracy, they remain using handcrafted metrics which require a large amount of manual work. More critically, models with handcrafted features have a poor generalization ability to cope with more complex and realistic data. Concerning this issue, Mi et al. [14] put forward deep learning techniques which enable neural networks to directly learn features correlated to code readability from the source code and achieved a state-of-the-art classification accuracy of 82.8%.

Though binary classification reaches a high performance, the practicability is still not strong. Because it is too rough to judge a code snippet as either readable or unreadable in practice especially for some snippets that are just neutral in readability. Thus, we propose to explore multi-class code readability classification which has better practicability in realistic scenarios. On top of that, considering the effectiveness of deep learning techniques and the limited dataset, our research extends the use of data augmentation and deep learning techniques onto multi-class classification.

III. Proposed Approach

We treat code readability classification as a multi-class classification task with three categories: readable, unreadable, and neutral. As shown in Figure 1, our proposed approach consists of three main steps.

A. Dataset Construction and Code Representation

1) Dataset Construction: Collected by conducting a large-scale survey to invite annotators for labeling code snippets, open-source datasets from Buse [5], Dorn [6] and Scalabrino [19] are usually used as ground-truth data by most code readability studies. The final readability score is the average of every annotator's rank. Different from the previous binary classification [5][6][13][14][15][19], we partition dataset into three readability categories based on the readability score assigned (five-point Likert scale) to support multi-class classification. In addition, we remove code snippets in other languages and use only code snippets in JAVA which is consistent with prior researches [14][15]. In total, there are 420 labeled JAVA code snippets that make up our dataset. The top 25% code snippets with the highest readable score are considered as readable whereas the bottom 25% is considered as unreadable [15]. Therefore, the middle 50% is treated as the neutral readability data. We belive this partition conforms to reality because highly readable or unreadable code is less common than neutral ones. We finally split the gathered dataset into a training dataset and a test dataset in the ratio of 8:2.

2) Code Representation: A proper code representation method is important for deep learning-based code readability classification. Mi et al. proposed and deeply discussed three representation methods that could effectively capture code readability-related information [13]. A series of experiments were conducted to evaluate which code representation method is the most effective. Whereas results reveal that character-level representation has an outstanding capability surpassing the other two methods (with classification accuracy of 88.0%, 81.0%, and 75.5% respectively). Therefore, we adopt character-level representation in this paper. Specifically, code snippets are treated as two-dimension character matrices in which every letter, number, mark, and whitespace is converted into its corresponding ASCII value.

B. Code Augmentation

Considering conducting a large-scale survey to label new code snippets is too costly, we decide to apply advanced data augmentation techniques to enlarge our dataset.

1) Domain-Specific Data Transformation Method : The feasibility of using domain-specific data transformation in binary code readability classification was preliminarily disclosed in a previous research [15]. Thus, we propose to transform and

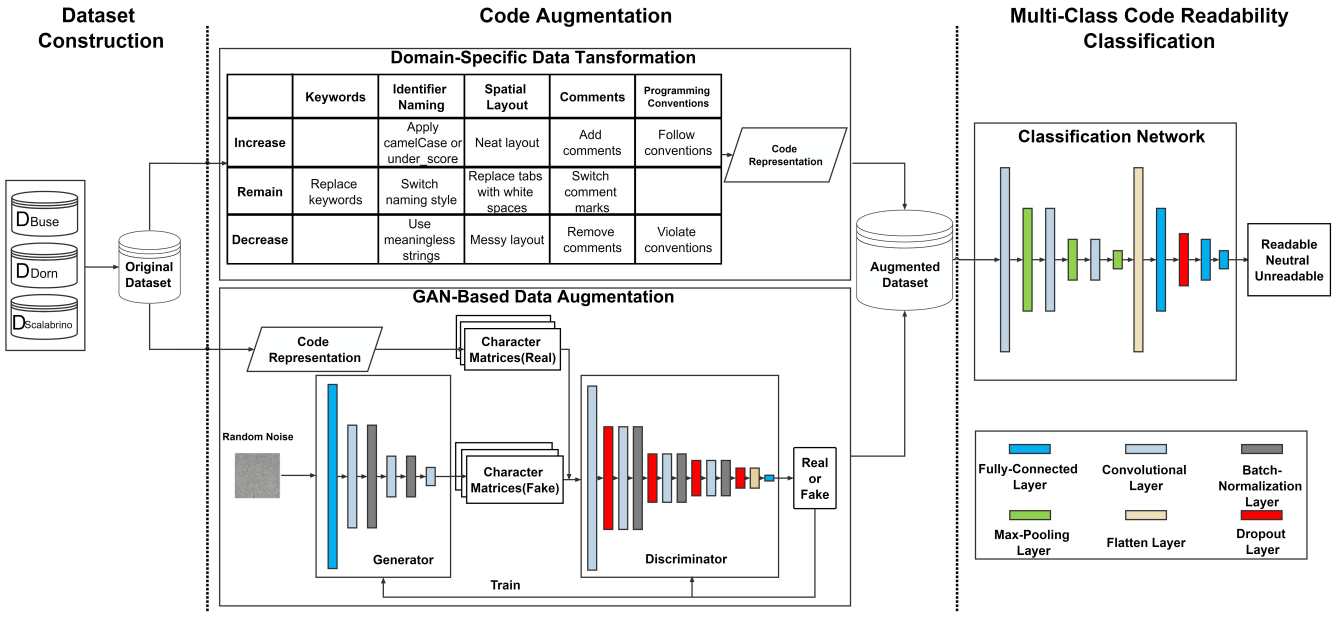


Fig. 1. Approach Overview

use it in multi-class classification. To adopt it in multi-class classification, we formulate three sets of operations as shown in Figure 1 that could generate new code snippets from the original ones without changing their label (which is the most secure way of augmentation because we cannot guarantee a correct output label after intentionally changing the original label). Specifically, we could perform increasing readability operations on readable data, decreasing readability operations on unreadable data, and remaining readability operations on neutral data to generate artificial data with the correct label.

2) GAN-Based Data Augmentation Method: Being able to generate artificial data out of a given dataset, GANs (i.e., Generative Adversarial Networks) have been proven to be a potent and efficient data augmentation method to compensate for the lack of data [3][7]. Specifically, we propose the use of the Wasserstein Generative Adversarial Network (i.e., WGAN) [2] for our task, because it could generate artificial data with a high diversity without the mode collapse and vanishing gradient problems [1].

Following the typical WGAN architecture, we construct our network as shown in Figure 1. The network is comprised of a generator and a discriminator. The generator could generate a character matrix from a given random noise, whereas the discriminator will determine if a given character matrix represents a real code snippet or a fake one generated by the generator. After adequate training, the generator should be able to generate verisimilar character matrices that could fool the discriminator and be treated as reliable artificially labeled data. To adopt it in our task, we put data with each readability label into training and generate new data with that label respectively. The detailed structures of the generator and the discriminator are introduced as follows.

- The generator starts with a fully-connected layer followed by two pairs of convolutional layers and batch-normalization layers. A convolutional layer is placed at the end. ReLU is set to be the activation function for all but not the last layer which uses Tanh as the activation. Furthermore, all convolutional layers use the same padding with the kernel size of 4 and all batch-normalization layers use the momentum of 0.8.
- The discriminator starts with a convolutional layer followed by three groups of dropout layers, convolutional layers, and batch-normalization layers. A dropout layer, a flatten layer, and a fully-connected layer are placed at the end of the discriminator. All convolutional layers use the same padding and the kernel size of 3 and all batch-normalization layers use the momentum of 0.8. The dropout ratio is set as 0.25. LeakyReLU with 0.2 as the alpha is used to be the activation function in this network.

During training the network, the loss function is Wasserstein Distance and the optimizer is RMSProp with the learning rate of 0.00005. We use the loss value to decide when to stop training. The output is scaled to integers in the range of -1 to 128 to get the same format as character matrices (see Section 3.1.2).

3) Parallel Augmentation Method: In this method, we use the two aforementioned methods, domain-specific data transformation and GAN-based data augmentation, to generate synthetic data separately, and then mix them to improve data diversity for training the classifier.

4) Sequential Augmentation Method: In this method, we first use domain-specific data transformation to generate synthetic data which is then used in the process of GAN-based data augmentation. After that, another batch of synthetic data is generated by GAN. Then, augmented data from both methods is mixed and used to train the classifier.

C. Multi-Class Classification Network

Considering the limited sample size, we propose a simple convolutional neural network [10]. The network starts with three pairs of convolutional layers and max-pooling layers. Then, there is a flatten layer followed by three fully-connected layers and a dropout layer as shown in Figure 1. RMS is used as the optimizer with a learning rate of 0.0015. Categorical cross-entropy is proposed to be the loss function [8].

IV. Experiment Setup

In this section, we present evaluation metrics and three research questions.

A. Evaluation Metrics

Considering that the number of code snippets varies in three readability levels, we propose to use the macro-accuracy and macro-f-measure, which are the most commonly used evaluation metrics in multi-class classification researches, to verify our experiment results. The evaluation metrics (Accuracy/Precision/Recall/F-measure) of different readability levels are directly added up for average, and all readability levels are given the same weight.

In addition, Brunner-Munzel test [4] is adopted as another evaluation metric to examine if there is a statistically significant difference between the results obtained with and without data augmentation. Furthermore, Cliff's δ effect size is used to quantify the magnitude of the measured difference.

B. Research Questions

Aiming to validate the effectiveness of our proposed approach, we formulate three research questions that will be answered through corresponding experiments. To improve generality, all experiments will be carried out in ten rounds.

RQ1: Which code augmentation method is the most effective for multi-class code readability classification?

Approach: We set the augmentation level[2] as N and 2N in this RQ because they are verified to be the most effective levels by the previous research [15]. Thus, we will compare the following four data augmentation methods with the augmentation levels of N and 2N:

- Domain-specific data transformation method
- GAN-based data augmentation method
- Parallel data augmentation method
- Sequential data augmentation method

RQ2: Which augmentation level is the best for multi-class code readability classification?

Approach: According to the result of RQ1, we would use the best data augmentation method to further probe the effect of different augmentation levels. Specifically, augmentation levels including 0N (no synthetic data), 1N, 2N, 3N, 4N, and 5N will be adopted and compared.

[2]Augmentation level stands for the ratio of augmented data to the original data, where N is defined as the total number of the original data

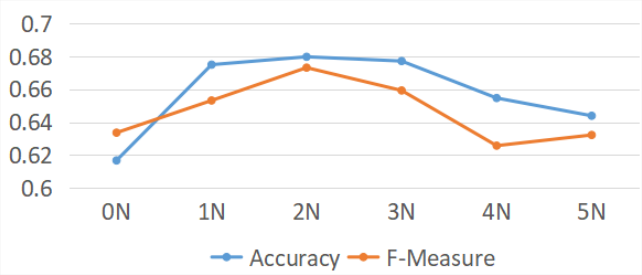


Fig. 2. Results of RQ2

RQ3: To what extent does data augmentation improve multi-class code readability classification?

Approach: Both the optimal augmentation method and level are used in this research question according to the results of RQ1 and RQ2. Instead of merely using the simple CNN (see Section 3.3) as the classification network, three other deep learning-based classifiers and two machine learning-based classifiers are chosen in order to explore the amount of improvement augmented data brought on different networks. In addition to accuracy and f-measure, the Brunner-Munzel test and Cliff's δ effect size [9] are also used in this RQ to provide a more intuitive and quantified evaluation revealing the improvement brought after the use of data augmentation.

V. Results

In this section, we present experimental results with respect to each RQ we proposed.

RQ1: Which code augmentation method is the most effective for multi-class code readability classification?

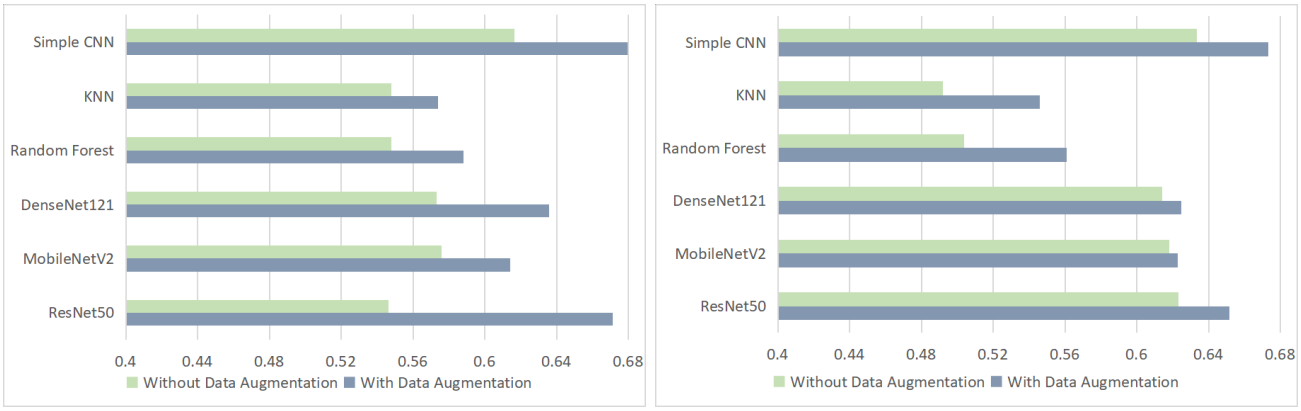
Based on the approach of RQ1, we repeat our experiments for ten rounds. The results are visualized in Table 1. It can be seen that the GAN-based data augmentation method outperforms the other three on both accuracy and f-measure. In contrast, the domain-specific data transformation method is comparatively inferior. We conjecture that snippets generated by the domain-specific data transformation method could not generalize beyond the original snippets which severely limits its performance. Whereas GAN could capture the readability features more accurately, thus improving the classifier performance. Besides, the two combined methods do not perform well. It might be due to data augmented by domain-specific transformation distracting the training of GAN which misinterprets the repeated parts as important and finally lower the quality of data generated by GAN. Considering that the GAN-based data augmentation method achieves a relatively better result in both evaluation metrics, we propose it to be the augmentation method used in RQ2 and RQ3.

RQ2: Which augmentation level is the best for multi-class code readability classification?

In this RQ, we evaluate the effect of different augmentation levels from 0N (i.e., only original data) to 5N. We conduct ten rounds of control experiments for each augmentation level.

TABLE I
RESULTS OF RQ1

Evaluation Metric	Without Augmentation	Domain-Specific Data Transformation	GAN-Based Data Augmentation	Parallel Data Augmentation	Sequential Data Augmentation
Accuracy	0.617	0.636 (1N) / 0.638 (2N)	0.675 (1N) / 0.680 (2N)	0.632 (1N) / 0.616 (2N)	0.623 (1N) / 0.658 (2N)
F-Measure	0.634	0.644 (1N) / 0.643 (2N)	0.653 (1N) / 0.673 (2N)	0.639 (1N) / 0.638 (2N)	0.625 (1N) / 0.659 (2N)



(a) Accuracy (b) F-Measure

Fig. 3. Results of RQ3

TABLE II
BRUNNER-MUNZEL TEST AND CLIFF'S δ EFFECT SIZE OBTAINED IN RQ3

Evaluation Metric	Simple CNN	KNN	Random Forest	DenseNet121	MobileNetV2	ResNet50
Brunner-Munzel test	0.00 (<0.05)	0.02 (<0.05)	0.00 (<0.05)	0.01 (<0.05)	0.15	0.00 (<0.05)
Cliff's δ effect size	0.68 (Large)	0.56 (Large)	0.84 (Large)	0.62 (Large)	0.38 (Medium)	0.83 (Large)

The results for accuracy and f-measure are shown in Figure 2. It can be observed that the optimal augmentation level is 2N in which the accuracy is 68.0% and the f-measure is 67.3%. Compared to the results where no augmented data is used, the improvements on accuracy and f-measure are significant with 6.3% and 3.9% respectively. However, a performance degradation appears when we increase the proportion of synthetic data over 2N. Thus, overusing artificially generated data is unhelpful and gives no further improvement. This conclusion falls in line with the prior paper [15], in which 1N and 2N are the best augmentation levels and there is also a decline in performance with too more synthetic data.

RQ3: To what extent does data augmentation improve multi-class code readability classification?

To comprehensively measure the improvement data augmentation brings, we select three other widely used deep learning-based classifiers, namely, RestNet50, DenseNet121, and MobileNetV2, and two machine learning-based classifiers, namely, random forest classifier and k-neighbors classifier. In terms of evaluation metrics, the Brunner-Munzel test and Cliff's δ effect size are also deployed to quantify the improvement along with accuracy and f-measure. Based on RQ1 and RQ2, we adopt the GAN-based data augmentation method with the augmentation level of 2N. Therefore, there are 1008 code snippets in the final training set and 84 code snippests in the final test set whereas the ratio of three readability labels remains 1:2:1 in both sets. The results are shown in Figure 3.

It is noticeable that data augmentation helps improve the performance of all classifiers. Because we did not fine-tune off-the-shelf deep learning-based models, they do not perform well than the simple CNN. The results of the p-value and the Brunner-Munzel test are shown in Table 2. It can be seen that all classifiers but MobileNetV2 have a p-value lower than 0.05 for the Brunner-Munzel test and a large d-value for Cliff's δ effect size. Both of them imply statistically significant improvements in classification accuracy.

VI. DISCUSSION

In this section, we discuss three worth noting aspects of our experiments.

Augmentation Effort. By adopting the data augmentation approach, the cost is largely reduced. The process of generating 5N (2100) artificial snippets by using the domain-specific data transformation was completed in two man-months. The process of generating data using GAN even saved more time and effort where it only took a day. Therefore, the data augmentation approach we proposed is both effective and efficient compared to the traditional way of conducting a large-scale survey to collect new data. However, such a survey is still necessary because all augmentation methods rely on the availability of ground-truth data.

Readability and Understandability. It is noticeable that the domain-specific data transformation does not perform well. We conjecture that it is due to the subjectivity code readability owns [5]. Because the entire transformation process is done by merely two people who might have a different

perspective on readability from the general people. Thus, data generated might have an inferior generality and limit the classifier's performance. Besides, manual manipulations might affect code understandability as well. Understandability is defined as to what extent the code allows developers to understand its purpose [11][20][22]. Readable code is not meant to be understood more easily. Thus, the domain-specific data transformation is only suitable for augmenting code readability data at the present stage.

Readability Data. The original dataset sourced from different prior researches [5][6][19] might be prone to errors and weaken our conclusions to a certain extent because three datasets are labeled by different groups of human annotators and differ remarkably in terms of code length, code completeness. For instance, the dataset collected by Scalabrino et al. is comprised of complete code snippets, whereas the other two datasets contain only partial snippets. Furthermore, the sample size might not be sufficient to train deep learning networks such as WGAN we used. As a result, GAN-based data augmentation could not be fully utilized and therefore produce ordinary performance. We believe that if the original dataset could be larger, the effectiveness of GAN-based data augmentation will be further improved. The shortage of data is also reflected in RQ3 where the simple CNN outperforms other complex networks due to over-fitting.

VII. Conclusions and Future Work

To enable multi-class code readability classification, we propose a data augmentation approach that could be used to effectively enlarge the dataset and well train a multi-class classifier. A series of experiments are conducted to evaluate the effectiveness of our proposed method. The results reveal that adopting the data augmentation approach could improve the classification performance to a considerable extent with the accuracy improved by 1.8% to 10.7%. The classifier produced by our proposed method could reach a state-of-the-art multi-class code readability classification accuracy of 68.0%, as well as 67.3% f-measure.

Our future work is mainly about improving the performance of the code readability classifier. We will try to utilize other code representation methods to capture more readability-related features such as code semantics. We will also improve the practicability of the code readability model. In fact, most text readability classification researches include more than three readability levels [12][16][17]. Therefore, we plan to increase the number of readability levels to be more realistic. Lastly, although considering data augmentation, the final size of the dataset is still too small for practical application because data augmentation highly relies on the original dataset. Therefore, labeling more data is still unavoidable to produce a high-performance code readability classifier.

Acknowledgments

This work was supported by the GHfund B (20220202, ghfund202202028015) and the Spark Project of Beijing University of Technology (Project No. XH-2021-02-24).

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] C. Bowles, L. Chen, R. Guerrero, P. Bentley, R. Gunn, A. Hammers, D. A. Dickie, M. V. Hernández, J. Wardlaw, and D. Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv preprint arXiv:1810.10863*, 2018.

[4] E. Brunner and U. Munzel. The nonparametric behrens-fisher problem: Asymptotic theory and a small-sample approximation. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 42(1):17–25, 2000.

[5] R. P. Buse and W. R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2009.

[6] J. Dorn. A general software readability model. *MCS Thesis available from (http://www. cs. virginia. edu/weimer/students/dorn-mcs-paper. pdf)*, 5:11–14, 2012.

[7] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[8] Y. Ho and S. Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2019.

[9] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, 22(2):579–630, 2017.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] J.-C. Lin and K.-C. Wu. A model for measuring software understandability. In *The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*, pages 192–192. IEEE, 2006.

[12] M. Martinc, S. Pollak, and M. Robnik-Šikonja. Supervised and unsupervised neural approaches to text readability. *Computational Linguistics*, 47(1):141–179, 2021.

[13] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao. Improving code readability classification using convolutional neural networks. *Information and Software Technology*, 104:60–71, 2018.

[14] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and X. Mei. An inception architecture-based model for improving code readability classification. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 139–144, 2018.

[15] Q. Mi, Y. Xiao, Z. Cai, and X. Jia. The effectiveness of data augmentation in code readability classification. *Information and Software Technology*, 129:106378, 2021.

[16] E. Miltsakaki and A. Troutt. Real time web text classification and analysis of reading difficulty. In *Proceedings of the third workshop on innovative use of NLP for building educational applications*, pages 89–97, 2008.

[17] M. K. Paasche-Orlow, H. A. Taylor, and F. L. Brancati. Readability standards for informed-consent forms as compared with actual readability. *New England journal of medicine*, 348(8):721–726, 2003.

[18] D. Posnett, A. Hindle, and P. Devanbu. A simpler model of software readability. pages 73–82, 2011.

[19] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto. Improving code readability models with textual features. pages 1–10, 2016.

[20] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically assessing code understandability: How far are we? In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 417–427. IEEE, 2017.

[21] T. Sedano. Code readability testing, an empirical study. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 111–117. IEEE, 2016.

[22] M.-A. Storey, K. Wong, and H. A. Müller. How do program understanding tools affect how programmers understand programs? *Science of Computer Programming*, 36(2-3):183–207, 2000.

[23] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2017.