An Inception Architecture-Based Model for Improving Code Readability Classification

Qing Mi, Jacky Keung, Yan Xiao, Solomon Mensah, Xiupei Mei
City University of Hong Kong, Kowloon, Hong Kong
{Qing.Mi, yanxiao6-c, smensah2-c, xpmei2-c}@my.cityu.edu.hk, Jacky.Keung@cityu.edu.hk

ABSTRACT

The process of classifying a piece of source code into a *Readable* or *Unreadable* class is referred to as *Code Readability Classification*. To build accurate classification models, existing studies focus on handcrafting features from different aspects that intuitively seem to correlate with code readability, and then exploring various machine learning algorithms based on the newly proposed features. On the contrary, our work opens up a new way to tackle the problem by using the technique of deep learning. Specifically, we propose IncepCRM, a novel model based on the Inception architecture that can learn multi-scale features automatically from source code with little manual intervention. We apply the information of human annotators as the auxiliary input for training IncepCRM and empirically verify the performance of IncepCRM on three publicly available datasets. The results show that: 1) Annotator information is beneficial for model performance as confirmed by robust statistical tests (i.e., the Brunner-Munzel test and Cliff's delta); 2) IncepCRM can achieve an improved accuracy against previously reported models across all datasets. The findings of our study confirm the feasibility and effectiveness of deep learning for code readability classification.

CCS CONCEPTS

• **Software and its engineering** → **Software reliability**; **Maintaining software**; • **Computing methodologies** → *Neural networks*; Supervised learning by classification;

KEYWORDS

Code Readability Classification, Inception Architecture, Deep Learning, Empirical Software Engineering

ACM Reference Format:
Qing Mi, Jacky Keung, Yan Xiao, Solomon Mensah, Xiupei Mei. 2018. An Inception Architecture-Based Model for Improving Code Readability Classification. In *EASE'18: 22nd International Conference on Evaluation and Assessment in Software Engineering, June 28–29, 2018, Christchurch, New Zealand.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3210459.3210473

1 INTRODUCTION

Reading source code is a frequent activity in software maintenance and evolution. Developers must read and fully understand source code before attempting any changes. It is therefore desirable to have *readable code* that is less erroneous [7], more reusable [23], and easier to maintain [27].

Code readability can be defined as a human judgment of how easily a piece of code can be read and understood [1], which is essentially an intuitive concept. As shown in Figure 1, the visual appearance of a source code in general is referred to as its readability [4]. If automated readability metrics were available, they could be integrated into real-world scenarios such as modern IDEs (integrated development environments) and version control products, and thus assist developers to identify poorly written code [1, 21]. However, there is only a limited literature concerning this issue.

```
/**
 * Add one zero if necessary
 * @param number
 * @return
 */
private CharSequence addZero(int number) {
  StringBuilder builder = new StringBuilder();

  if (number < 10) {
    builder.append('0');
  }

  builder.append(Integer.toString(number));
```

Readable Code (Mean Readability Score - 4.02)

```
  /**
   * Creates a new <code>DisbandUnitAction</code>.
   *
   * @param freeColClient The main controller object for the client.
   */
  DisbandUnitAction(FreeColClient freeColClient) {
      super(freeColClient, "unit.state.8", null, KeyStroke...
      putValue(BUTTON_IMAGE, freeColClient.getImageLibrary()...
      ImageLibrary.UNIT_BUTTON_DISBAND,
              0));
      putValue(BUTTON_ROLLOVER_IMAGE, freeColClient...
      getUnitButtonImageIcon(
              ImageLibrary.UNIT_BUTTON_DISBAND, 1));
```

Unreadable Code (Mean Readability Score - 2.01)

Figure 1: Readable Code vs. Unreadable Code

The existing code readability studies mainly consist of three phases: 1) Collect labeled data by conducting a large-scale survey, inviting multiple human annotators to rate code snippets by readability; 2) Handcraft features from different aspects that intuitively seem to have some effect on code readability; 3) Train a machine learning classifier using data collected from the first phase.

Unlike prior work [1, 7, 21, 23], we propose IncepCRM, a deep learning based model for code readability classification. Specifically, we make three improvements: 1) We eliminate the need for manual feature engineering; 2) We apply the information of human annotators as the auxiliary input for model training; 3) We propose a modified Inception module that can automatically learn multi-scale

EASE'18, June 28–29, 2018, Christchurch, New Zealand

ACM ISBN 978-1-4503-6403-4/18/06…$15.00
https://doi.org/10.1145/3210459.3210473

features from the source code. The experimental results show that IncepCRM performs better than previous approaches, achieving a state-of-the-art accuracy ranging from 82.2% to 84.2%.

The rest of this paper is organized as follows. Section 2 presents the background and related work. Section 3 introduces IncepCRM in detail. Section 4 describes the design of our empirical study and Section 5 discusses the results. We analyze threats to validity in Section 6. Section 7 concludes the paper and suggests future work.

2 BACKGROUND AND RELATED WORK

This section introduces the related work on code readability research and the basic knowledge of Convolutional Neural Networks.

2.1 Code Readability Research

A wide variety of readability metrics for natural languages are available in the literature (e.g., the Flesch-Kincaid Grade Level [8] and the SMOG Grading [19]), yet only a few studies have considered the counterpart targeted in the context of program source code.

With data collected from 120 human annotators, Buse et al. [1] proposed the first model of code readability based on a simple set of local code features (e.g., the number of identifiers). Buse et al.'s work was a major contribution to this area because it opened up the possibility of automated readability metrics and provided an excellent platform for conducting future readability experiments. Posnett et al. [21] improved Buse et al.'s work by introducing a simple, intuitive theory of readability, which relied on two main measures: size and code entropy. They showed that the simple model outperformed Buse et al.'s model across all performance measures (e.g., F-Measure and ROC). In order to build a model of code readability that is more likely to generalize than previous work, Dorn et al. [7] performed a large-scale human study involving over 5000 participants and proposed structural pattern features, visual perception features, alignment features, and natural language features. In a recent study, Scalabrino et al. [23] further introduced a set of textual features that were based on source code lexicon analysis. The results indicated that the proposed features complemented classic structural features when predicting code readability judgments.

As we have described above, most existing work still relies on manual feature engineering. In other words, they focus on handcrafting different combinations of surface-level features to represent code readability. The process is generally effort-intensive and requires strong domain-specific knowledge [5]. Actually, code readability is an intuitive concept, the manually-designed features are likely to be inadequate and thus put significant limitations on the model performance. In order to eliminate the need for manual feature engineering and increase the accuracy of code readability classification, we propose to employ Convolutional Neural Networks which are more efficient in learning local higher-level features of source code [5].

2.2 Convolutional Neural Network

Deep learning [14] is a subfield of machine learning that is inspired by Artificial Neural Networks, which has achieved impressive (and often state-of-the-art) results in various domains such as image recognition [10, 24] and natural language processing [3, 13]. Inspired by the great success, deep learning has attracted considerable attention of researchers and practitioners in software engineering community. For instance, Gu et al. [9] applied the RNN Encoder-Decoder model to generate API usage sequences for a given natural language query. Wang et al. [28] leveraged Deep Belief Network to learn a semantic representation of programs and used the learned features to improve defect prediction. However, few studies have been reported to apply deep learning techniques on code readability classification.

Convolutional Neural Network (ConvNet) [16] is one of the most popular deep learning models. A typical ConvNet is composed of stacked convolutional, pooling, and fully-connected layers. As an example, consider LeNet-5 [15, 26] shown in Figure 2, which is designed for handwritten digit recognition.

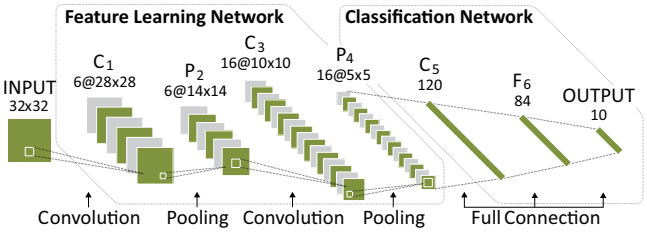

Figure 2: Architecture of LeNet-5 (A Classical Convolutional Neural Network)

As shown in Figure 2, a typical ConvNet consists of two parts. The first part is usually a feature learning network, which includes alternating convolutional and pooling layers. A convolutional layer is used to extract features from local receptive fields. A feature map (the planes in Figure 2) is obtained by applying a convolution operation to the input followed by a non-linear activation function. Specifically, the k^{th} feature map at a given layer is generated by:

$$h^k = \sigma(W^k * x + b^k) \tag{1}$$

where W^k denotes the weight matrix, b^k denotes the bias unit, and $\sigma(\cdot)$ denotes a non-linear activation function (e.g., Tanh ($\frac{e^x - e^{-x}}{e^x + e^{-x}}$) or ReLU ($max(0, x)$). Generally, each hidden layer is composed of multiple feature maps: $\{h^k, k = 0...K\}$. After a convolutional layer, a pooling layer is added to perform down-sampling and reduce computation complexity by consolidating the features learned in the previous layer. The second part of ConvNets is usually a classification network containing one or more fully-connected layers, which is used to generate the output based on the extracted features.

3 PROPOSED APPROACH

Analogous to prior work [1, 7, 21, 23], we regard the code readability classification problem as a binary classification problem: given a piece of source code, we classify it as either *Readable* or *Unreadable*.

To improve the accuracy of code readability classification, we propose IncepCRM, a deep learning based method that can extract features automatically from source code with little manual intervention. Figure 3 presents the overall workflow of IncepCRM.[1] In

[1]Note that the human study (code readability survey) is used to obtain a reliable oracle, i.e., a set of labeled data for model training. The process is shown here for clarity, which is not part of our approach.

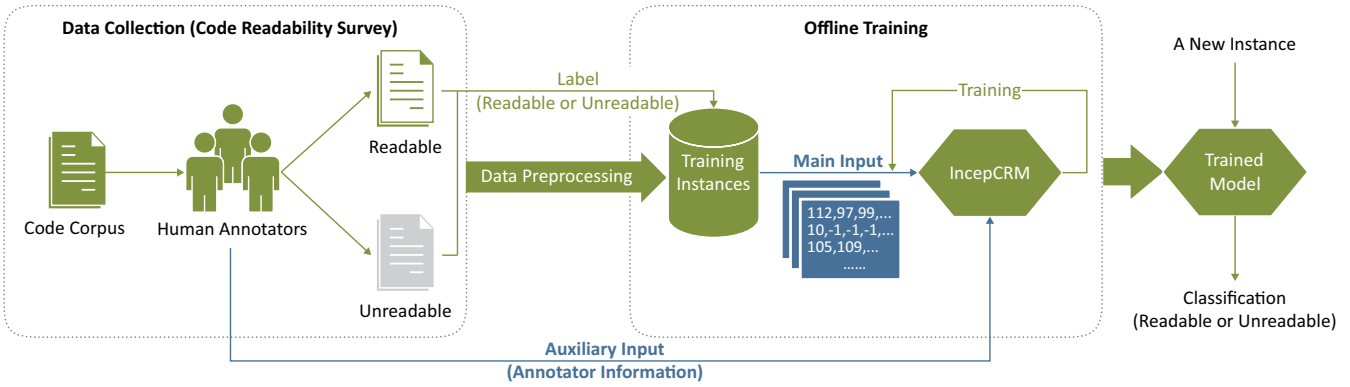

Figure 3: Approach Overview

the first step, we transform source files (data collected from the code readability survey) into a specific format that Convolutional Neural Networks (ConvNets) can accept. Inspired by GoogLeNet [26], we build a novel model based on the Inception architecture for the task of code readability classification. After that, we involve annotator information as the auxiliary input for model training. Using the trained model, we are able to predict the classification for each new instance. A detailed description of our approach is provided in the following sections.

3.1 Data Preprocessing

Based on various reasons, previous studies on deep learning applications in software engineering often preserve partial information of the source code. For instance, Wang et al. [28] made use of only three types of AST nodes (i.e., invocation nodes, declaration nodes, and control-flow nodes) for defect prediction. Gu et al. [9] focused solely on API usage sequences (e.g., *FileOutputStream.new*, *FileOutputStream.write*, and *FileOutputStream.close*) for API recommendation. By contrast, we intend to preserve the original information of the source code as much as possible, primarily because no definitive conclusions have been reached as to which factor(s) can affect code readability.

Inspired by image recognition, we propose to treat a source code as a matrix of characters (similar to pixels of an image). Specifically, we first convert letters (i.e., a-z and A-Z), numbers (i.e., 0-9), marks (e.g., parentheses and braces), and whitespaces (i.e., spaces, horizontal tabs, and line terminators) into their ASCII values. Because ConvNets require all inputs to be of the same length, we then pad the matrices with a special integer (here, -1). In comparison to existing code representations (e.g., AST nodes), the main advantage of this method is its simplicity and without loss of generality.

3.2 Annotator Information

According to the definition of code readability, the most accurate method to judge whether a piece of source code is readable is to conduct a large-scale survey, inviting multiple human annotators (preferably domain experts) to provide readability scores based on their knowledge and experience, as shown in the first part of Figure 3. The result of the survey is a set of code snippets accompanied by readability scores, which serves as the main input to IncepCRM.

Considering that the obtained mean-opinion-scores may not be equally reliable, we propose to apply the information of human annotators as the auxiliary input to our model. Although a variety of metrics can be considered, in this study we especially focus on the following two:[2] 1) the number of annotators who have rated a code snippet (code snippets that have been rated by more annotators are more reliable than those with less); 2) the standard deviation of readability scores on a code snippet (a lower standard deviation indicates greater consistency among annotators).

3.3 Model Architecture

Inspired by GoogLeNet [26],[3] we adapt the Inception architecture for code readability classification. The primary reasons are as follows: 1) The Inception architecture is proven to be effective in capturing multi-scale information from the input data; 2) When building a ConvNet layer, there is a challenge of choosing either a convolution or a pooling operation. The Inception architecture enables the simultaneous use of both.

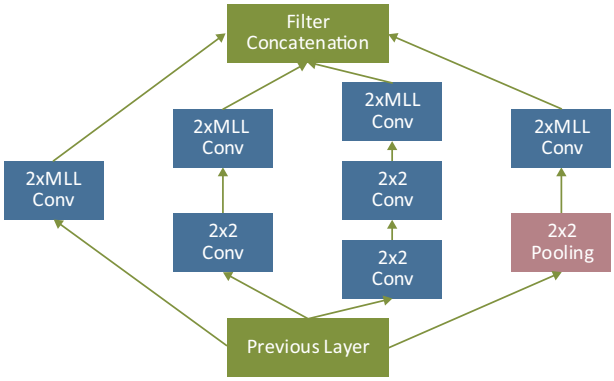

Figure 4: The Inception Module used in IncepCRM

Figure 4 illustrates the Inception module we use in IncepCRM, which contains a set of convolutions and poolings alongside each

[2]The main reason is that the information of human annotators has not been properly recorded by prior work [1, 7, 23]. As a result, we have only limited data to work with.
[3]GoogLeNet is a deep ConvNet constructed by stacking Inception modules.

Table 1: Statistical Summary of the Code Corpus

Dataset	Description	Source	Total # of Code Snippets	Total # of Annotators	Lines of Code[a]	
					Mean	SD
D_{Buse}	Provided by Buse et al. [1]	SourceForge	100	121	7.61	2.59
D_{Dorn}	Provided by Dorn et al. [7]	SourceForge	360	5000+	29.74	16.36
$D_{Scalabrino}$	Provided by Scalabrino et al. [23]	SourceForge	200	9	26.56	10.29

[a] All lines in the text of source code are counted, including blank and comment lines.

other. Specifically, we make two modifications based on Inception-v6 [25]. First, we apply the same receptive field in each module for computational savings. We bias our selection to minimal values (here, 2) to capture lower-level features. Second, because code readability is context sensitive (just like natural language), the width of the filter size on top levels is deliberately set as the maximum line length (MLL).

In IncepCRM, the feature learning network consists of Inception modules, whereas the classification network consists of fully-connected layers followed by a Softmax layer. To introduce non-linearity and accelerate the training process, ReLU activation and batch normalization are applied after each convolutional layer. In addition, we involve the dropout strategy to reduce over-fitting.

3.4 Model Training

We use the Keras library[4] with Tensorflow backend to implement IncepCRM. The training process is to minimize the loss (the classification error) of the network by adjusting the layer parameters (i.e., weights and biases) in an iterative way. Here, we adopt the cross-entropy loss [6] (one of the most commonly used loss functions for classification problems) to calculate the error rate between actual and desired outputs. The loss function is then given by:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} p_j^{(i)} \log q_j^{(i)} \tag{2}$$

where $N = 32$ is the number of data samples (the batch size), $M = 2$ is the number of classes (*Readable* and *Unreadable*), p_j denotes whether data sample i belongs to class j (the ground-truth), and q_j denotes the probability for class j estimated by IncepCRM.

We train the model in a supervised manner using the Adam optimizer [11] with a learning rate of 0.001. The number of feature maps is set to 32 and the dropout rate is set to 0.5.[5] The training data comes from three publicly available datasets, including a total of 210 code snippets with a shape of 50×305 (the details are provided in Section 4.1). For the model to generalize well, we split the data into two parts: 70% for training and 30% for test.

4 EMPIRICAL STUDY DESIGN

We aim to answer the following research questions:

- **RQ1:** Does annotator information contribute to the performance of IncepCRM?
- **RQ2:** To what extent does IncepCRM outperform the existing models in code readability classification?

We detail our research settings in this section. Specifically, we first describe the construction process of the studied datasets. Then we briefly introduce the evaluation metrics used in this study. Finally, we present our research methodology and competitors according to each research question.

4.1 Data Preparation

To train and test our model, we first need to prepare a corpus of labeled code snippets as the ground-truth. To facilitate comparison with other approaches, we use publicly available data provided by previous studies [1, 7, 23], namely, D_{Buse}, D_{Dorn}, and $D_{Scalabrino}$. Table 1 presents a statistical summary of the code corpus.

In D_{Buse}, D_{Dorn}, and $D_{Scalabrino}$, each code snippet is evaluated by multiple human annotators using a Likert scale [18] ranging from 1 (very unreadable) to 5 (very readable). To label a code snippet as *Readable* or *Unreadable*, we follow a similar approach to that of Lee et al. [17]. First, we aggregate the readability scores of each code snippet by taking the mean. After that, we select the top 25% of code snippets (with high readability scores) as the *Readable* group, whereas the bottom 25% of code snippets (with low readability scores) as the *Unreadable* group. By doing so, we obtain two representative groups for each dataset.

4.2 Measures for Evaluation

We adopt accuracy to evaluate classification results, which is a widely used performance measure in the context of code readability classification [1, 7, 23]. Accuracy shows the ratio of all correctly classified instances, which is defined as:

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{3}$$

where *tp*, *fp*, *tn*, *fn* are the number of readable instances that are correctly classified as *Readable*, the number of unreadable instances that are wrongly classified as *Readable*, the number of unreadable instances that are correctly classified as *Unreadable*, and the number of readable instances that are wrongly classified as *Unreadable*, respectively. The higher the metric value, the better the model performance.

4.3 Analysis Method

To answer RQ1, we examine whether there exists significant difference between the performance of IncepCRM with and without annotator information using robust test statistics recommended

[4] https://github.com/keras-team/keras

[5] As a result of time constraint, the values of hyperparameters (e.g., learning rate and dropout rate) are chosen arbitrarily. In future work, we will fine-tune these hyperparameters in a systematic manner to improve our model.

by Kitchenham et al. [12], i.e., the Brunner-Munzel test (significance level $\alpha = 0.05$) [20] and Cliff's delta [2]. The value of Cliff's delta is interpreted using the thresholds provided in Romano et al.'s study [22], which is considered *Negligible* for $|d\text{-}value| < 0.147$, *Small* for $|d\text{-}value| < 0.330$, *Medium* for $|d\text{-}value| < 0.474$, and *Large* for otherwise.

To answer RQ2, we compare IncepCRM with five state-of-the-art code readability models:

- **Buse et al.'s Model [1]:** To construct an automated model of readability, Buse et al. proposed a relatively simple set of low-level code features (e.g., the number of loops).
- **Posnett et al.'s Model [21]:** Based on Buse et al.'s work, Posnett et al. derived a model that is simpler, better performing, and theoretically well-founded with only 3 variables: $z = 8.87 - 0.033V + 0.40Lines - 1.5Entropy$.
- **Dorn et al.'s Model [7]:** Dorn et al. presented a formal descriptive model of code readability based on features related to the visual and linguistic presentation of code.
- **Scalabrino et al.'s Model [23]:** By considering lexical aspects of source code involved in identifiers and comments, Scalabrino et al. further introduced a set of textual features (e.g., narrow meaning identifiers).
- **A Comprehensive Model [23]:** A code readability model which combines all the features mentioned above.

We abbreviate the aforementioned models as M_{Buse}, $M_{Posnett}$, M_{Dorn}, $M_{Scalabrino}$, and M_{All}, respectively. To enable a fair comparison, we apply different machine learning techniques as the underlying classifier for each model, namely, Logistic Regression (LR), Bayesian Network (BN), Multilayer Perception (MLP), Sequential Minimal Optimization (SMO), and Random Forest (RF).

5 RESULTS AND DISCUSSION

In this section, we present the results of our empirical study with respect to the two research questions.

RQ1: Does annotator information contribute to the performance of IncepCRM?

This RQ is to investigate whether the addition of annotator information can further improve the accuracy of code readability classification. We observe that without annotator information, IncepCRM is able to correctly classify 80.19% of code snippets in our corpus. When annotator information is included, the accuracy increases to 82.76%. The *p-value* for the Brunner-Munzel test is 0.016 (<0.05) and the *d-value* for Cliff's delta is 0.641 (>0.474), implying a significant difference with a large effect size.

Our work is the first attempt to explore the impact of annotator information in the context of code readability classification. However, we involve only two metrics. In future work, we plan to examine additional factors, for instance, the annotators' development experience (the opinions of domain experts are more reliable than those of university students).

RQ2: To what extent does IncepCRM outperform the existing models in code readability classification?

This RQ is to explore the extent to which IncepCRM can predict human readability judgments compared to previously reported models in the literature. We provide the accuracy achieved by each model on three publicly available datasets in Table 2. The results show that IncepCRM yields higher accuracy against its competitors across all datasets.

Table 2: Accuracy Achieved by Code Readability Models

Model		D_{Buse}	D_{Dorn}	$D_{Scalabrino}$
M_{Buse}	LR	0.810	0.786	0.705
	BN	0.760	0.750	0.625
	MLP	0.760	0.742	0.665
	SMO	0.820	0.797	0.670
	RF	0.800	0.781	0.680
$M_{Posnett}$	LR	0.780	0.728	0.660
	BN	0.760	0.681	0.695
	MLP	0.770	0.703	0.655
	SMO	0.770	0.711	0.680
	RF	0.770	0.703	0.605
M_{Dorn}	LR	0.800	0.800	0.755
	BN	0.670	0.747	0.640
	MLP	0.720	0.725	0.685
	SMO	0.770	0.767	0.725
	RF	0.750	0.744	0.690
$M_{Scalabrino}$	LR	0.740	0.772	0.680
	BN	0.530	0.681	0.640
	MLP	0.770	0.742	0.665
	SMO	0.720	0.717	0.650
	RF	0.720	0.742	0.655
M_{All}	LR	0.790	0.839	0.795
	BN	0.720	0.761	0.710
	MLP	0.730	0.778	0.700
	SMO	0.810	0.806	0.770
	RF	0.770	0.789	0.690
IncepCRM		**0.825**	**0.842**	**0.822**

There are three main reasons why IncepCRM can surpass the state-of-the-art. First, we have eliminated the need for manual feature engineering, and thus effectively avoided personal biases and neglects of certain features. Second, a modified Inception architecture has been specially designed to learn a high-level representation of the source code. Finally, the visual appearance of code plays a critical role in its readability (see Figure 1), while deep learning has achieved remarkable success in image-related tasks. However, it should be noted that deep learning is not flawless. The main drawback lays in the lack of interpretability of the obtained model.

6 THREATS TO VALIDITY

Internal Validity. Our code corpus is sourced from prior work [1, 7, 23]. If the data is prone to errors, then it will be reflected in our results. Further research is needed to explore this issue.

Another possible threat resides in the sample size. According to Table 1, only a few labeled data are available in the literature, which may not be adequate to train the proposed IncepCRM. To mitigate this threat, we intend to extend the code corpus with more samples in our future work.

External Validity. All projects investigated in this study are collected from SourceForge[6]. It is unclear whether the same findings would be observed in other projects. To improve external validity, we plan to experiment our model on other open source projects to better generalize our results.

Construct Validity. In our experimental setup, we mainly used accuracy to evaluate the effectiveness of the proposed model. However, other performance measures such as F-Measure and AUC can also be considered. We leave this for future studies.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we presented IncepCRM, a novel approach to improve the accuracy of code readability classification based on the Inception architecture. Specifically, we made the following contributions:

- We proposed a modified Inception module suitable for code readability classification.
- We proposed the use of annotator information as the auxiliary input for model training. To the best of our knowledge, we are the first to consider the effect of human factors.

To validate our approach, we conducted a series of experiments on three publicly available datasets. The results show that: (RQ1) Annotator information is beneficial for classification performance; (RQ2) IncepCRM achieves state-of-the-art accuracy in comparison to previous code readability models [1, 7, 23].

Our future work focuses on further improving the performance of IncepCRM. For instance, we can fine-tune hyperparameters (e.g., filter size and dropout rate), combine multiple deep learning models, or experiment with different configurations (e.g., different optimizers and loss functions).

Additionally, there is a need to find out what exact factors make a code more or less readable. We plan to further explore this open question with the help of deep learning techniques. Once completed, we are able to provide targeted suggestions to developers on how to improve their code.

ACKNOWLEDGMENTS

This work is supported in part by the GRF of the Research Grants Council of Hong Kong [No. 11208017], and the research funds of City University of Hong Kong [No. 9678149, 9440180, 7004683, and 7004474].

REFERENCES

[1] Raymond P L Buse and Westley R. Weimer. 2010. Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering* 36, 4 (jul 2010), 546–558. https://doi.org/10.1109/TSE.2009.70

[2] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509. https://doi.org/10.1037/0033-2909.114.3.494

[3] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2017. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Vol. 1. 1107–1116.

[4] Ermira Daka, José Campos, Gordon Fraser, Jonathan Dorn, and Westley Weimer. 2015. Modeling readability to improve unit tests. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, New York, New York, USA, 107–118. https://doi.org/10.1145/2786805.2786838

[5] Hoa Khanh Dam, Truyen Tran, John Grundy, and Aditya Ghose. 2016. DeepSoft: a vision for a deep model of software. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, Vol. 1691. ACM Press, New York, New York, USA, 944–947. https://doi.org/10.1145/2950290.2983985 arXiv:1602.05561

[6] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. 2005. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research* 134, 1 (feb 2005), 19–67. https://doi.org/10.1007/s10479-005-5724-z

[7] Jonathan Dorn. 2012. A General Software Readability Model. *MCS Thesis avairable from (http://www. cs. virginia. edu/˜ weimer/students/dorn-mcs-paper. pdf)* (2012).

[8] Rudolph Flesch. 1948. A new readability yardstick. *Journal of applied psychology* 32, 3 (1948), 221.

[9] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*. ACM Press, New York, New York, USA, 631–642. https://doi.org/10.1145/2950290.2950334 arXiv:1508.06655

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 770–778. https://doi.org/10.1109/CVPR.2016.90 arXiv:1512.03385

[11] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. (2014), 1–15. https://doi.org/10.1145/1830483.1830503 arXiv:1412.6980

[12] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (apr 2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[13] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48. PMLR, 1378–1387. http://proceedings.mlr.press/v48/kumar16.html

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444. https://doi.org/10.1038/nature14539 arXiv:1312.6184v5

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1, 4 (dec 1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[16] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. 396–404.

[17] Taek Lee, Jung Been Lee, and Hoh Peter In. 2013. A study of different coding styles affecting code readability. *International Journal of Software Engineering and its Applications* 7, 5 (2013), 413–422. https://doi.org/10.14257/ijseia.2013.7.5.36

[18] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[19] G Harry Mc Laughlin. 1969. SMOG grading-a new readability formula. *Journal of reading* 12, 8 (1969), 639–646.

[20] Karin Neubert and Edgar Brunner. 2007. A studentized permutation test for the non-parametric Behrens-Fisher problem. *Computational Statistics & Data Analysis* 51, 10 (jun 2007), 5192–5204. https://doi.org/10.1016/j.csda.2006.05.024

[21] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A simpler model of software readability. In *Proceeding of the 8th working conference on Mining software repositories - MSR '11*, Vol. 11. ACM Press, New York, New York, USA, 73. https://doi.org/10.1145/1985441.1985454

[22] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the NSSE and other surveys. In *annual meeting of the Florida Association of Institutional Research*. 1–33.

[23] Simone Scalabrino, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. 2016. Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, Vol. 2016-July. IEEE, 1–10. https://doi.org/10.1109/ICPC.2016.7503707

[24] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Science (New York, N.Y.)* 313, 5786 (sep 2014), 504–7. https://doi.org/10.1126/science.1127647 arXiv:1409.1556

[25] Christian Szegedy. [n. d.]. Scene classification with inception-7.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1–9. https://doi.org/10.1109/CVPR.2015.7298594 arXiv:1409.4842

[27] Yahya Tashtoush, Zeinab Odat, Izzat Alsmadi, and Maryan Yatim. 2013. Impact of Programming Features on Code Readability. *International Journal of Software Engineering and Its Applications* 7, 6 (nov 2013), 441–458. https://doi.org/10.14257/ijseia.2013.7.6.38

[28] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, Vol. 14-22-May-. ACM Press, New York, New York, USA, 297–308. https://doi.org/10.1145/2884781.2884804

[6] https://sourceforge.net