# USING MACHINE LEARNING IMAGE RECOGNITION FOR CODE REVIEWS

Michael Dorin[1, 2], Trang Le[2],
Rajkumar Kolakaluri[2] and Sergio Montenegro[1]

[1]Aerospace Information Technology, Universität Würzburg,
Würzburg, Germany
[2]Engineering, University of St. Thomas, St. Paul, MN, USA

***ABSTRACT***

*It is commonly understood that code reviews are a cost-effective way of finding faults early in the development cycle. However, many modern software developers are too busy to do them. Skipping code reviews means a loss of opportunity to detect expensive faults prior to software release. Software engineers can be pushed in many directions and reviewing code is very often considered an undesirable task, especially when time is wasted reviewing programs that are not ready. In this study, we wish to ascertain the potential for using machine learning and image recognition to detect immature software source code prior to a review. We show that it is possible to use machine learning to detect software problems visually and allow code reviews to focus on application details. The results are promising and are an indication that further research could be valuable.*

***KEYWORDS***

*Code Reviews, Machine Learning, Imagine Recognition, Coding Style*

## 1. INTRODUCTION

Code reviews are policy in many software development organizations, and it is commonly believed that code reviews are an economical way to discover faults before a software product is deployed. Indeed, it is even suggested that code that has not been adequately reviewed has twice the faults of reviewed code [1]. However, many software engineers are overwhelmed with work, so proper code reviews are often not done. The reviewability of software is affected by many factors such as documentation, logic, semantics, and syntax. Source code includes aspects that might even be considered aesthetic, and aesthetic aspects might turn tedious and possibly overwhelm the review process [2]. In a paper by Yazdani and Manovich, non-photographic images, such as screenshots and images of text messages, were analysed and found they could be useful in predicting social trends [3]. This paper aims to evaluate the possibility of using "screenshots" of source code with machine learning image recognition as part of the software code review process. Tools to reduce monotonous tasks related to reviews could be very valuable. This paper begins by discussing the readability aspects of code and estimates the impact style has on reviews. We then created images of poorly styled code and properly styled code and used machine learning to train an image recognizer to identify poorly formatted code and present positive results. Creating source code “screenshot images” for analysis could be part of automating code reviews. Using automation as part of the review process could make software engineers more efficient.

David C. Wyld et al. (Eds): ACITY, DPPR, VLSI, WeST, DSA, CNDC, IoTE, AIAA, NLPTA - 2020
pp. 159-167, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101514

# 2. RELATED WORK

## 2.1. Code Reviews

As code reviews are an essential topic, many papers are written each year to address the review process's problems. In the paper, "Confusion in Code Reviews," the authors recognize that code reviews do not always go smoothly and identify items confusion in the review process [4]. Fatima et al. discuss the good and bad consequences of feedback in the review process [5]. A vital feature these papers discuss is related to problems of the code review process, and by automating some of the toils of the review process, we believe it is possible to improve the overall quality of the review.

## 2.2. Machine Learning and Image Analysis

Machine Learning and Image Recognition has been used with success in many areas. For example, Lin et al. describe the successful use of deep learning for laser positioning [6]. An even more applicable subject is image processing and sentiment analysis. Qian et al. analysed twitter messages attempting to capture human expressiveness with image recognition [7]. Zhang et al. (2015) describe microblogs' sentiment analysis by integrating text and image features [8]. Although these papers were not software related, they positively demonstrate the success of machine learning in the context of image analysis as well as showing the possibility of detecting text sentiment.

## 2.3. Machine Learning and Source Code

Concerning research related directly to software source code, the paper, "Aesthetics Versus Entropy in Source Code," found that evaluating code beauty could be used for style checking [9]. Other studies have used machine learning and deep learning in code review systems to analyse code errors automatically. Bielki et al. introduced a machine learning-based system where the analyser learned to produce static analysis tools using a decision tree algorithm [10]. The system showed a coverage improvement but mentioned scalability and generalizability could be improved. Gupta and Sundaresan created a system using a 'long short-term memory' network called DeepCodeReviewer, which learned to review from human reviews. The authors explain in their paper they plan to improve the DeepCodeReviewer tool to learn continuously and personalize itself to a team or a repository [11]. These papers demonstrate the applicability of machine learning to the code review process, but do not address reviews using image processing.

# 3. BACKGROUND FOR METHODS

## 3.1. Data from Previous Work

This paper uses some data originally gathered in preparation for the 2019 IEEE Aerospace Conference in Big Sky, Montana (Aeroconf). At Aeroconf, we wanted to study what stylistic issues were most problematic for code reviews [7]. For this study, we created 'code snippets' and asked programmers to determine the proper outcome should the code be executed. This survey demonstrated that problematic code not only takes longer to review, but it is more often reviewed incorrectly. The survey showed that improperly formatted code had a review success of less than 90% on average, and on average it took about 22.5 seconds longer to review than properly formatted code. Some feedback received from this presentation indicated that many issues could be avoided simply by following coding standard rules. We agree, as in general, the issues

identified were stylistic, not logic-based. With this in mind we suggest these issues may be spotted visually, analogous to a tumour in a medical CT scan.

## 3.2. Scope of the Problem

Even though modern code editors can enforce properly formatted code, we were still surprised to see how much existing code violates style rules. It seems that even though modern tool kits are helpful, some issues of poorly formatted code linger. To demonstrate the ramifications of this problem, we downloaded several hundred projects from GitHub and scanned them for the common issues. As shown in Table 1, we discovered that most projects had at least some software issues and two projects had more than 15% of their lines associated with a issues. Figure 1 shows this table graphically. We used static analysis tools 'nsiqcppstyle'[12] and 'lizard'[13] to identify issues.

Table 1.

| Percent | Value |
|---|---|
| 0 to 1% | 180 |
| 1.1% to 2% | 181 |
| 2.1% to 3% | 121 |
| 3.1% to 4% | 51 |
| 4.1% to 5% | 36 |
| 5.1% to 6% | 23 |
| 6.1% to 9% | 22 |
| 9.1% to 12% | 15 |
| 12.1% to 15% | 3 |
| More than 15% | 2 |
| Increased Time | 21% |

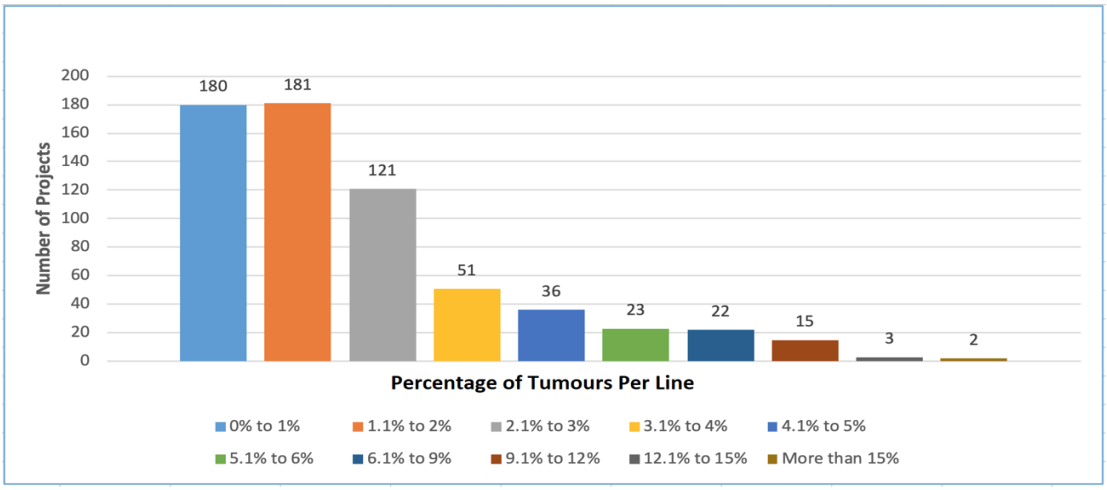


Figure 1. Percentage of Lines Associated with Tumours

## 3.3. Impact of Tumours

We will examine a hypothetical project to illustrate the possible consequences of tumours in source code. To arrive at a reasonable size for our hypothetical project, we analysed projects in our collection with very few tumours, specifically the projects where 0% to 2% of their lines were associated with a tumour, as shown in Figure 1. Static analysis of this group showed the median number of lines was 61,649, and the median number of tumours was 499, with the

probability of a line being part of a tumour being 0.008. Our hypothetical project was given characteristics based off these numbers and is shown in Table 2 with the results. We used the results of the Aeroconf [7] study to estimate how long code with and without tumours takes to review, and in our hypothetical project the presence of tumours increased the review time by 21%. It is also important to note that not only does the presence of tumours increase the review time, but it also reduces the accuracy of the review to lower than 90% [7].

Table 2. Hypothetical Project Specifications

| **Description** | **Value** |
|---|---|
| Project Lines | 61,649 |
| Project Tumours | 499 |
| Code Segments Lines | 56 |
| Segment Count | 1,100 |
| No TumourSegment Review Time | 37.5 Seconds |
| Segment Review Time with Tumour | 59.5 Seconds |
| **No Tumours Review Time** | **11.5 Hours** |
| **Tumour Review Time** | **14.5 Hours** |
| **Increased Time** | **21%** |

### 3.4. Deep Learning Review

In recent years, deep learning and convolutional neural networks (CNN) have been shown to be efficient in resolving complex non-linear problems and have become a prevalent approach for a broad range of tasks [14]. By automating the process of feature learning, CNN takes advantage of the concept of local information and effectively detects different deep features in multiple successive stacked layers [15]. This has resulted in CNN becoming one of the most popular methods for image recognition and classification. In this paper, we leveraged the powerful capacities of CNN to develop our recognition system.

VGG-19 and ResNet50 are state-of-the-art convolutional neural networks that are trained on the ImageNet dataset for solving image classification tasks in computer vision [15][16]. VGG-19 is a classic convolutional neural network that has 19 layers with trainable weights, in which exists 16 Convolutional layers and 3 fully connected layers [17]. ResNet50 (Residual Networks) has 50 layers [16]. They are both applied to solve tasks related to transfer learning. These networks are used for this study as they possess a rich feature representation and can likely yield good results.

## 4. METHODS AND ALGORITHMS

### 4.1. Identification of the Tumours

As previously mentioned, hundreds of GitHub projects were selected in order to conduct the analysis [18]. Problematic code was identified through static analysis and image files were created. Good code snippets were likewise identified and separated, and the resulting image files were also produced. The process for image creation is shown in Figure 2. In total, a set of about 44,000 images were prepared and collected for model training and testing, with about 38,000 images labelled as non-tumour and the remaining labelled as tumour (about 6,000). However, due

to the imbalanced distribution ofdata, only about 6,000 of the non-tumour images were randomly selected, along with the 6,000 images of tumour data. Doing this, the model potentially avoids the problem of overfitting, which is a major factor leading to the poor performance of a model and boosts its overall performance significantly.

Figure 3. Example Source Code Image

The created images are monochromatic and 228 pixels by 280 pixels in size. An example is shown in Figure 3.

## 4.2. Implementation

In training the models, a Keras framework [19][20] is utilized with a Tesla T4 GPU [21]. All the models are rigorously trained over 100 epochs. Data is broken down into batches of 64 images. Different optimizers were tried to accelerate the training process, and stochastic gradient descent was selected with a learning rate of 0.001. Also, ImageDataGenerator [22] is employed to progressively manipulate and load data in batches and help monitor the training process.

As shown in Figure 4, ImageDataGenerator implements multiple stages in a single platform, ranging from the preprocessing of data, the code-snippets in this case, to the training and evaluating the resulting models. Using ImageDataGenerator makes the training and testing straightforward, and it also helps effectively monitor the training. We choose this class as an intermediary is because of the automation and convenience it offers. ImageDataGenerator is a very beneficial tool and is heavily used for big datasets demonstrating training automation and computational efficiency.

In this project, two advanced architectures for image classification, VGG-19 [23] and ResNet50 [24], as well as a customized CNN, were adopted. We felt that traditional machine learning methods would be ineffectual in capturing spatial and local features in images. Accordingly, we adopted the advancements of deep neural networks as the principal proposition of this study. In particular, VGG19 and ResNet50 were trained on top of two fully connected layers interwoven with a dropout layer and a pooling layer in between.

Transfer learning is regularly used with pre-trained models as a starting point for further development on the task at hand. Consequently, we exploited transfer learning to utilize the well-tuned pre-defined architectures to accelerate the result and speed up the training. With minor modifications on this predictive task, pre-trained models can harness the proven feature learning capacity and yield better outcomes.

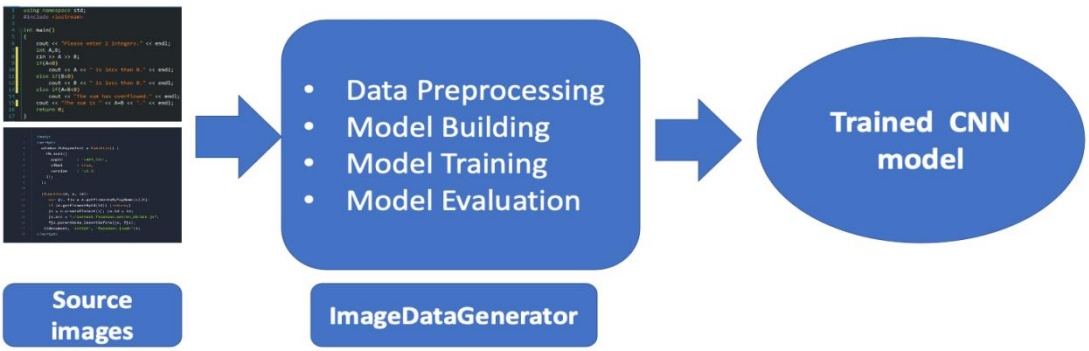


Figure 4. Pre-processing Data and Training Models

The customized CNN consists of three convolutional blocks which are followed by two fully connected layers at the end for classifying whether a given snippet of source code contains a tumour or not. By training the network, the model will be learning the weights and adjusting according to the training process that the model went through.

All the models are trained on the same data for the sake of accuracy rate comparison. The model selection is executed manually after the training and the model which achieved the best result in model evaluation phase is selected as a final model for predictions (Figure 5).

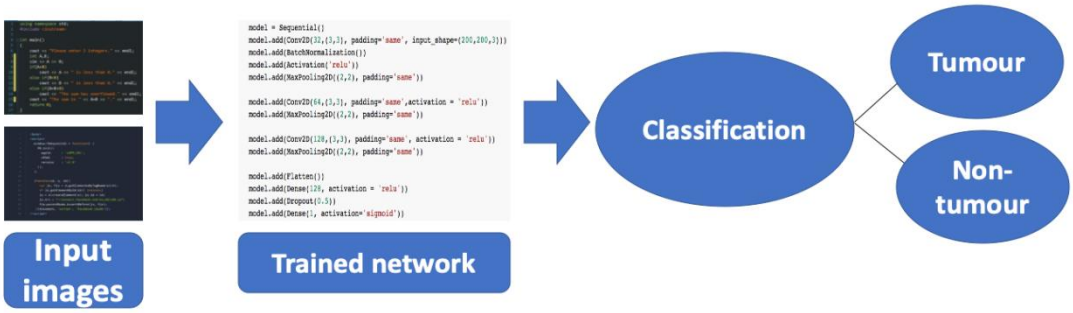


Figure 5. Model Evaluation

## 5. RESULTS AND DISCUSSION

To evaluate performance of the architectures, two metrics were used: accuracy and F1-score. Accuracy is adopted to measure how accurate the models are and F1-score is also elected so that the way the models make predictions can be observed and accessed more closely. We tried to balance the distribution of tumour and no-tumour training images, yet the probability of predicted images still might not be well distributed, so steps were taken to help mitigate noises and over fitting. The customized CNN architecture was found to be the best model with 80% accuracy and a 0.79 F1-score. VGG-19 and ResNet50 did not seem to perform as well on this task with results of only 59% and 55% respectively in terms of accuracy. Relevant comparisons are shown in Table 3.The problem might lie in the inherent nature and the amount of data presented. The

application of transfer learning makes a smooth transition to solve the problem, but the models do not align well with the data leading to the performances being weak. In addition, the amount of training data is not sufficient to bear the number of layers that constructs the notable distinction of the two state-of-the-art architectures. Our customized CNN has fewer layers and a less complicated architecture then ResNet50 and VGG-19. Also, since this is a binary classification problem, it seems to fit the data better than other pretrained powerful models such ResNet50 or VGG-19.

Table 3. Model Accuracy Rate Comparison

| **Description** | **Accuracy** | **F1-Score** |
|---|---|---|
| VGG19 | 59% | 0.62 |
| ResNet50 | 55% | 0.61 |
| Our CNN | 80% | 0.79 |

As we performed a reduction in training data earlier, class distribution over the data is balanced, and hence the problem of overfitting was avoided. This is exhibited via the results in the confusion matrix shown in Table 4. The number of correctly predicted tumours is appropriate for the number of tumours exists in the dataset. In other words, the precision and recall rates of the target classes are relevant and consistent, which demonstrate the efficiency of the model.

The deep convolutional network designed for this project achieved an impressive outcome as compared to VGG-19 and Resnet50. Using a convolutional model, 80% accuracy was achieved in detecting software tumours in the code snippets, which shows it is possible for a CNN to identify software segments with code tumours.

Even though we obtained a satisfying outcome, there are still gaps that could be filled in order to achieve a better effect. On the one hand, data is essential for any CNN model to operate well on a given task. Therefore, with improved tools to extract and process data, we could expect the model to perform better, and accordingly gain more effect. On the other hand, a larger and more sophisticated network can be employed to learn more complex features with the introduction of both spatial and temporal information into the network. These topics should be addressed in future research.

Table 4. Classification Report of the Final Model

| **Description** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|
| No tumour | 0.77 | 0.85 | 0.81 |
| Tumour | 0.83 | 0.74 | 0.78 |
| **Macro average** | 0.81 | 0.80 | 0.80 |
| **Weighted Average** | 0.81 | 0.80 | 0.80 |

## 6. CONCLUSION

Even for humans, visually recognizing a code tumour in software is a difficult task. Applying deep learning image classification brings a huge advancement and a positive outcome. This paper shows it is possible to use a convolutional neural network with up to 80% accuracy to identify patterns in code. Though our work identifies patterns that might be spotted by static analysis tools, it is possible that other tumour styles can be identified and discovered by our method. It seems it is possible to identify any bad looking code using machine learning and image recognition. Other methods should certainly be explored and researched since this work shows promising results, and tools can be created to enhance code review practices and perhaps other aspects of software development. Further research is required to make this system practical and useful.

## ACKNOWLEDGMENTS

Special thanks to the University of St. Thomas for supporting this work.

## REFERENCES

[1] Bavota, G. & Russo, B. (2015). "Four eyes are better than two: On the impact of code reviews onsoftware quality," 2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 81–90. doi:10.1109/ICSM.2015.7332454

[2] Kozbelt, A. ; Dexter, S.; Dolese, M.; Seidel, A. (2012). "The Aesthetics of Software Code: A quantitative exploration," Psychology of Aesthetics, Creativity, and the Arts, Vol. 6, No. 1, 57–65. doi:10.1037/a0025426

[3] Yazdani, M.& Manovich, L. (2015). "Predicting social trends from non-photographic images on Twitter," Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015, Institute of Electrical and Electronics Engineers Inc., 1653–1660. doi:10.1109/BigData.2015.7363935

[4] Ebert, F., Castor, F., Novielli, N., &Serebrenik, A. (2019). "Confusion in code reviews: Reasons, impacts, and coping strategies," In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 49-60). IEEE.

[5] Fatima, N.; Nazir, S.; Chuprat, S. (2020). "Understanding the Impact of Feedback on Knowledge Sharing in Modern Code Review," Institute of Electrical and Electronics Engineers (IEEE), 1–5. doi:10.1109/icetas48360.2019.9117268

[6] Lin, C.-S.; Huang, Y.-C.; Chen, S.-H.; Hsu, Y.-L.; Lin, Y.-C. (2018). "The Application of Deep Learning and Image Processing Technology in Laser Positioning," Applied Sciences, Vol. 8, No. 9, 1542. doi:10.3390/app8091542

[7] Dorin, M. & Montenegro, S. (2019). "Eliminating Software Caused Mission Failures," IEEE Aerospace Conference Proceedings,IEEE Computer Society. doi:10.1109/AERO.2019.8741837

[8] Zhang, Y.; Shang, L.; Jia, X. (2015). "Sentiment analysis on microblogging by integrating text and image features," Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 9078), Springer Verlag, 52–63. doi:10.1007/978-3-319-18032-8_5

[9] Coleman, R. & Boldt, B. (2017). "Aesthetics versus entropy in source code," In Proceedings of the International Conference on Software Engineering Research and Practice (SERP) (pp. 113-119). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[10] Bielik, P.; Raychev, V.; Vechev, M. (2016). "Learning a Static Analyzer from Data," Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 10426 LNCS, 233–253

[11] Gupta, A. &Sundaresan, N. (2018). "Intelligent code reviews using deep learning," doi:10.1145/nnnnnnn.nnnnnnn

[12] Yoon, J & Kunal, T. (2014). "Cpp style checker in python," From https://github.com/kunaltyagi/nsiqcppstyle, accessed 21-7-2020

[13] Yin,T. (2012). "A simple code complexity analyser without caring about the C/C++ header files or Java imports, supports most of the popular languages," fromhttps://github.com/terryyin/lizard, accessed 21-7-2020

[14] Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J.; Chen, T. (2018). "Recent advances in convolutional neural networks, Pattern Recognition, Vol. 77, 354–377. doi:10.1016/j.patcog.2017.10.013

[15] Shin, H. C.; Roth, H. R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R. M. (2016). "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning," IEEE Transactions on Medical Imaging, Vol. 35, No. 5, 1285–1298. doi:10.1109/TMI.2016.2528162

[16] Akiba, T., Suzuki, S., & Fukuda, K. (2017). "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," arXiv preprint arXiv:1711.04325.

[17] Mateen, M.; Wen, J.; Nasrullah; Song, S.; Huang, Z. (2018). "Fundus Image Classification Using VGG-19 Architecture with PCA and SVD," Symmetry, Vol. 11, No. 1, 1. doi:10.3390/sym11010001

[18] GitHub (2020). "The world's leading software development platform," from https://github.com/, accessed 21-7-2020
[19] Keras (2020). "Keras: the Python deep learning API," from https://keras.io/, accessed 21-7-2020
[20] Manaswi, N. K. &Manaswi, N. K. (2018). "Understanding and Working with Keras, Deep Learning with Applications Using Python," Apress, 31–43. doi:10.1007/978-1-4842-3516-4_2
[21] NVIDIA (2020). "NVIDIA T4 Tensor Core GPU for AI Inference | NVIDIA Data Center," from https://www.nvidia.com/en-us/data-center/tesla-t4/, accessed 21-7-2020
[22] Karis (2020). "Image data preprocessing," from https://keras.io/api/preprocessing/image/, accessed 21-7-2020
[23] He, K.; Zhang, X.; Ren, S.; Sun, J. (2016). "Deep residual learning for image recognition," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 2016-December), IEEE Computer Society, 770–778. doi:10.1109/CVPR.2016.90
[24] Simonyan, K. & Zisserman, A. (2014). "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556.