Deep learning-based software energy consumption profiling

Preprint · April 2019

CITATIONS
0

READS
441

1 author:

Muhammed Maruf Ozturk
Suleyman Demirel University/Engineering Faculty
45 PUBLICATIONS **170** CITATIONS

SEE PROFILE

Deep learning-based software energy consumption profiling*

Muhammed Maruf Öztürk

Suleyman Demirel University, Isparta, Turkey
muhammedozturk@sdu.edu.tr

Abstract. Energy efficient software development is a requirement to meet software quality standards. A great number of works have been done to enhance the level of information related to software energy consumption (SEC). They are generally focused on raw code data. These data can be profiled to predict SEC trends of future versions of a software. However, SEC works lack energy profiling with powerful predictive models. In this work, a deep learning-based SEC model is proposed. The model is than evaluated with 14 open-source projects. The experiment shows that deep learning performs better in SEC profiling than the alternatives such as random forest. Further, contrary to expectations, the success of the profiler is sensitive for the number of hidden layers of deep neural network.

Keywords: Software engineering · Energy consumption model · Green software · Deep learning.

1 Introduction

In recent years, practitioners doing experiment on software development have become aware of the importance of green software [23]. In accordance with this awareness, they began to consider green software requirements in software development [3]. Inherently, advance in mobile application increased the number of works related to SEC [16–18, 25]. However, evolution of software systems have led to an expansion in the scope of SEC studies. For instance, embedded and object-oriented software systems are the examples of this expansion [14].

Software has a critical role in energy consumption in terms of hardware level [13]. To reduce energy consumption of hardware resources, component-oriented methods are developed. They can be focused on CPU, network, and data storage. However, the decline in SEC is not sufficient without changing the behaviour of a software. Thus, maximum decline has been detected three percent so far. In order to increase this rate, alternative methods are required.

A software consists of various versions of them. To obtain a better insight into the software quality, SEC should be included in the quality evaluation criteria. Observing SEC in evolving software versions could give useful tips for software

* Supported by Intel.

developers with respect to green software. To the best of our knowledge, the works related to SEC are generally focused on refactoring of software development models [22]. However there is a need for profiling SEC with reliable models to predict future trends in energy consumption rates.

Predictive models are constructed with common learners such as random forest, naive bayes, and support vector machine [2]. However, with the great computation ability of new generation computers, deep neural network (DNN) has become widespread among researchers developing predictive models [19], [4], [24], [8]. DNN are used in various fields such as geometric problems, big data, classification, and image processing. In particular, the competitive advantage of DNN has seen in forecasting accuracy [19].

In this work, a deep learning-based SEC profiling method is proposed. The method is then tested on 14 open-source object-oriented projects. According to the obtained results; 1) Proposed method shows competitive performance in terms of energy consumption profiling, 2) The number of hidden layers of DNN is not directly proportional to the accuracy of profiling, 3) Large-scale software systems are much more suitable for establishing DNN-based models.

The rest of the paper is organized as follows. Section 2 mentions related works. The method proposed in this work is in Section 3. The tools utilized to establish the experiment, preliminaries of the method, and data sets are also detailed in this section. Experimental findings are given in Section 4. Section 5 concludes the results.

2 Related work

The main objective of this section is to give a historical summary about DNN along with selected works.

Primitive computer systems had not coped with multiple-layer structure in DNN due to the insufficient computational capability. However, in the end of 2009, learner based on DNN was introduced. One of the first designed DNN methods was for auido classification [15]. In subsequent years, some works pointed out that DNN would not be limited with academic research [1]. They have contributed greatly to understanding the underlying mechanism of DNN methods [9].

DNN studies are twofold. First, some of them aim at developing new techniques to improve DNN in terms of theoretical. They generally propose modification on DNN structure. Second, DNN is applied on a specific field such as speech recognition and image classification. In the beginning, deep belief networks (DBN) were widespread. DNN has a feedforward network structure along with directed connections in the layers. In contrast, DBN has undirected connection between layers and they are generally trained with unsupervised methods that accelerates the learning mechanism.

Lee et al. [15] used DBN to classify audio data. Their method achieved higy accuracy in TIMIT data set with a convolutional DBN. Model uncertainty problem is well known among researchers handling with Bayesian approximation. Gal

and Ghahramani investigated Bayesian and deep learning models together to cast dropout training in DNN [10]. They concluded that using DNN in Bayesian models helps reduce RMSE value in training. Chen et al. discussed DNN in terms of big data along with brief description of current issues [5]. Their work was cited by a great number of works including a big data experiment. DNN was also used in the classification of hyperspectral data. In this context, a DNN classifier namely SAE-LR [6] was proposed. Inspite of disadvantageous training time, SAE-LR has completed testing faster than other competitive methods such as SVM and KNN. Medical image processing is another application area of DNN [11]. In such works, having noise-free data is of great importance as much as developing novel methods producing high sensitivity and specificity [20].

With respect to the software engineering, deep learning found various subfields for itself such as software repository, code clone detection, fixing code errors, and vulnerability detection. White et al. [27] showed that deep learning algorithms enhance the quality of software language modeling. They also stated that using DNN to optimize hyperparameters of machine learning methods used for software engineering tasks could mitigate well-known search-based problems. DNN is useful for detecting similar patterns. For instance, a DNN method [26] was developed for code clone detection. It is capable of pairing file and method level components. Code errors can be fixed with the help of DNN. A DNN method namley DeepFix [12] was proposed by Gupta et al. to fix c errors. DeepFix was able to fix 27% of codes completely and 19% codes partially. It has an iterative repair design. Deep language models could help predict sequences of codes. Such a work was performed by Dam et al. [7] to predict long-term dependencies of software code. They achieved 4.7% performance improvement in training. One of the most profiling-focused studies was performed by Romansky et al. [21]. In their work, DNN was utilized to create time series of SEC. However, they did not consider the correlation between source codes.

3 Method

SEC is simply measured by $P = ExT$. However, this measurement brings some requirements. First, the machine in which SEC is recorded should be evaluated in terms of idle case. Considering such a case helps differentiate real consumption from all SEC including idle and individual rates. CPU cycle counts give tips for SEC of a software, and therefore, the tools developed for measuring SEC generally takes useful information from system records including CPU cycle. In the method, Intel Power Gadget version 3.0 has been used for recording SEC. Intel Power Gadget is able to record valuable information related to SEC such as CPU power consumption (watt), temperature of CPU, and frequency values. SEC of processor cores and graphics are called IA and GT energy, respectively. SEC can be represented with Joules and mWh in Intel Power Gadget. Figure 1 is an example of Intel Power Gadget logging screen. It was produced by executing two YouTube social media videos simultaneously for 254 second. In this figure, SEC details is given with processor energy consumption in terms of milliwatt.

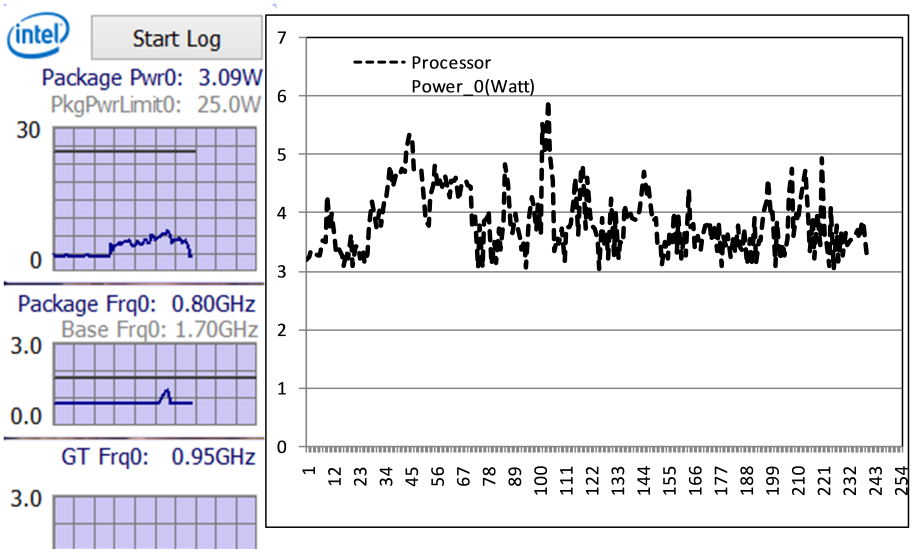

Fig. 1: Intel Power gadget logging screen.

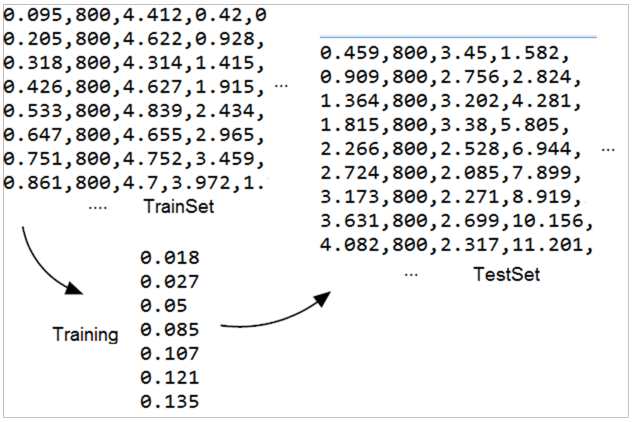

Fig. 2: Training and testing process of a data set produced by Intel Power Gadget. Training data is trained with recorded SEC rates. Subsequently, SEC values of testing data are predicted.

Deep learning codes have been designed with *deepnet* which is a DNN package of R. *deepnet* provides user to implement DNN architecture. 17 features of the data sets are employed for profiling. In the beginning, all the data is exposed to normalization. To predict SEC trends, 5 applications are used to create training data sets. Trained DNN is then tested on 1 application. In these processes, equal number of the features in training and testing should be used. Prediction accuracy is also investigated in terms of the number of hidden layers.

In train function, default settings of *dbn.dnn.train* function is used (learningrate=0.8, activefunction=sigmoid, numepoch=3, batchsize=100, momentum=0.5). The format of the data used for profiling is given in Figure 2.

3.1 Deep learning

SEC data recorded during the experiment is matrix format. If a matrix has n features, a vector x in matrix is as $x = 1...n$. In any matrix, $x_1, ..., x_m$ represents all the instances. This matrix is used as input layer in a DNN. Any hidden layer is denoted with h and $h_1, .., h_m$ includes all hidden layers. If we have t hidden layers, weighted hidden layers are denoted with $h^1, .., h^t$. For each hidden output, Equation 1 is employed. In this equation, m represents the number of features and w represents related weight. A multi-layer neural network is presented in Figure 11.

$$W.X = w_1.x_1 + w_2.x_2 + ... + w_m.x_m \tag{1}$$

In a multi-layer neural network, it is decided whether a neuron should be activated or not. This operation called activation function suppresses noise points to help network get useful information. Activation function selected in a DNN could be linear, sigmoid, or scaled version of sigmoid called tanh. In Figure 4, the active function used in DNN model is shown. The formula of Figure 4 is given in Equation 2. In this function, the output becomes 1 if related sigmoid neuron produces 0.5 or higher output value. Otherwise, it outputs 0. If x is 0, any set up rule can not be established. x in Equation 2 can be calculated via Equation 3. Here p denotes input data and w is the related weight.

$$f(x) = 1/1 + e^{-x} \tag{2}$$

$$x = \sum_{i=1}^{m} w_i.p_i \tag{3}$$

3.2 Data sets

To perform the experiment, 14 open source C# projects were retrieved from cyotek and GitHub that they include a great number of open-source projects for developers and researchers. Details of the data sets are given in Table 2. The metrics extracted during the experiment is seen in Table 1 Having a wide variety of projects may have extended the generality of the experiment.

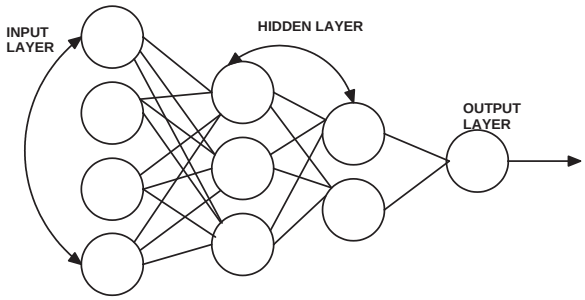

Fig. 3: A multi-layer neural network.

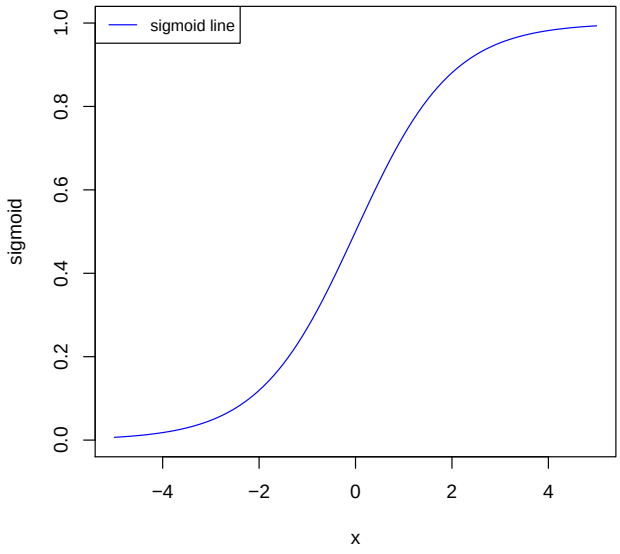

Fig. 4: Active function of hidden unit. In the experiment, sigmoid type is selected.

Table 1: Metric details.

Name	Description
LOC	Lines of code
BLOC	Blank Lines of Code
SLOC-P	Physical Executable Lines of Code
SLOC-L	Logical Executable Lines of Code
MVG	McCabe VG Complexity
C&SLOC	Code and Comment Lines of Code
CLOC	Comment Only Lines of Code
CWORD	Commentary Words
HCLOC	Header Comment Lines of Code
HCWORD	Header Commentary Words

Table 2: Details of the projects used in the experiment.

Project	LOC	BLOC	SLOC-P	SLOC-L	MVG	C&SLOC	CLOC	CWORD	HCLOC	HCWORD
cmyk	1728	241	1246	886	25	1	241	612	0	8
awesome-dotnet-core	1131	216	859	615	37	3	56	281	0	0
colorDistance	888	148	627	420	20	0	113	249	0	0
corelDraw	2839	364	2231	973	98	1	244	833	0	11
defectFramework	185186	11788	109126	71171	13610	19	64272	274843	235	1430
drawAnimated	331	54	231	127	9	3	46	226	0	15
getText	660	126	463	341	13	0	71	256	0	0
imageBoxTiff	849	86	614	440	8	0	149	289	0	0
listBoxEmpty	450	80	303	167	6	4	67	275	0	24
langTonSim	3932	479	2924	1984	115	7	529	1096	0	1
quadEquation	956	100	626	435	20	4	230	884	10	40
riffPalette	772	133	514	323	22	2	125	470	0	8
sourceSafe	4154	640	3048	1932	216	5	466	1539	0	0
wavePlayResult	370	33	273	193	9	0	64	197	0	0

Table 3: Fomulas of performance parameters.

name	**formula**
True positive rate sensitivity recall	TP/(TP+FN)
False positive rate	FP/(FP+TN)
Precision	TP/(TP+FP)
True negative rate Specificity	TN/(TN+FP)
F-measure	$\frac{(2*Recall*Precision)}{(Recall+Precision)}$
Accuracy	$\frac{(TN+TP)}{(TN+FN+FP+TP)}$
G-mean	$\sqrt{sensitivity.specificity}$

4 Results

Performance measures used in the experiment are given in Table 3. Energy consumption rates of colorDist and getText are seen in Figure 8 and 7, respectively. In a program like colorDist, SEC increases at a stable rate in the beginning of the application. For idle state, SEC dos not increase rapidly. Re-executing the application with some commands lead to an increase as in the beginning. getText is a program which enters an infinitive loop from the beginning of the application. Such an execution produces SEC graph as in Figure 7. getText has a lower

complexity than colorDistance. It can be concluded that SEC lines of software systems having high complexity become unstable. Figure 5 illustrates profiling performed with 200 random instances. The prediction completed with Random Forest does not perform well especially for the instances 40-60. On the other hand, DNN has an higher accuracy than Random Forest as seen from Figure 6.

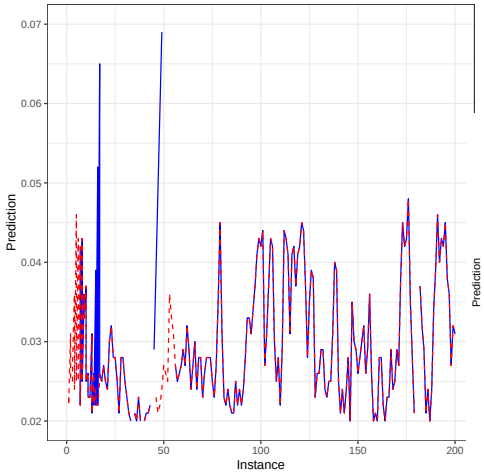

Fig. 5: SEC profiling with random forest predictor. Random instances retrieved from all projects. Real and predicted SEC are represented with red and blue lines, respectively.

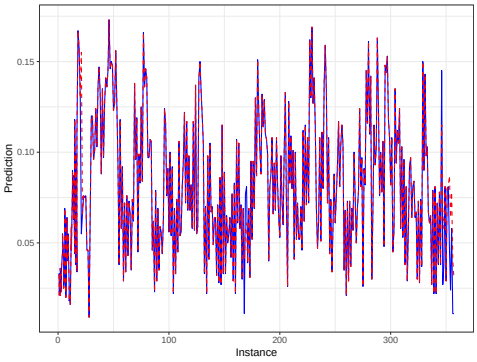

Fig. 6: SEC profiling with DNN. Random instances retrieved from all projects. Real and predicted SEC are represented with red and blue lines, respectively.

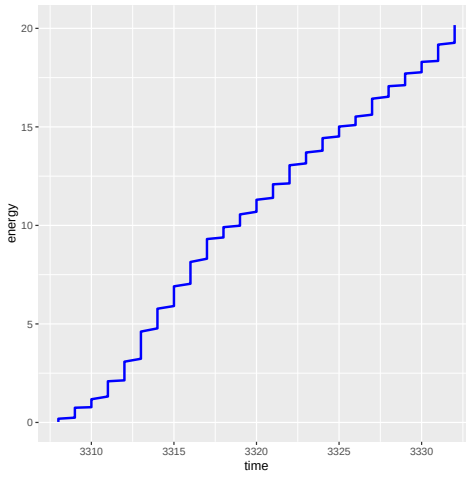

Fig. 7: Cumulative energy consumption rate (joule) of getText. Y axis represents related seconds.

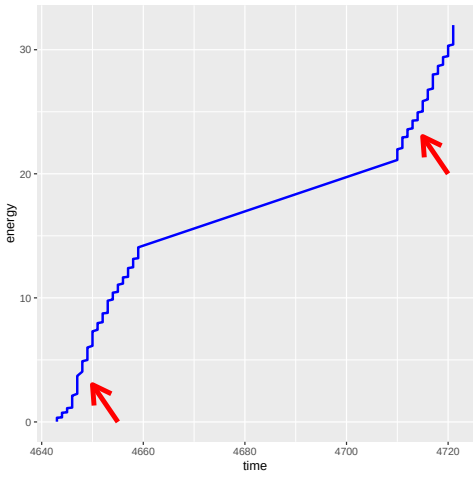

Fig. 8: Cumulative energy consumption rate (joule) of colorDistance. Y axis represents related seconds.

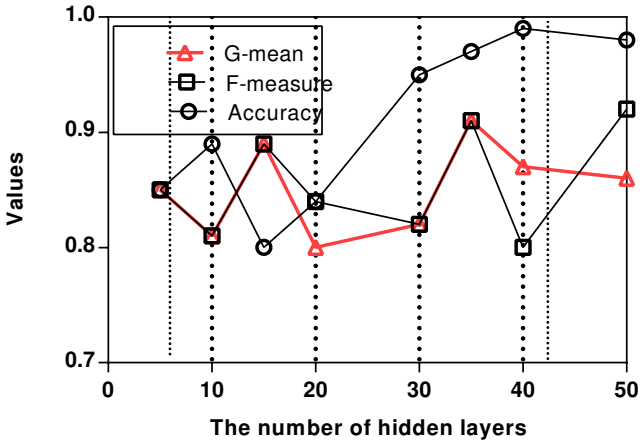

Fig. 9: Profiling performance with respect to three performance parameters.

5 Conclusion

This paper proposes a SEC profiling method based on DNN. The method is then tested on 14 open-source projects along with Random Forest. According to the obtained results, DNN gives higher accuracy than Random Forest. Further,

the number of the hidden layers of DNN is crucial for creating reliable models. The accuracy of the profiling is not directly proportional with the number of hidden layers (Figure 9) that can be determined depending on the type and the scale of the data sets. In the experiment, default configuration of *deepnet* package was used. For instance, sigmoid was used for activation function. Tuning methods may be investigated to go further to increase the success of the profiling. Last, the method presented in this paper should be examined in terms of mobile applications.

References

1. Arel, I., Rose, D.C., Karnowski, T.P., et al.: Deep machine learning-a new frontier in artificial intelligence research. IEEE computational intelligence magazine **5**(4), 13–18 (2010)
2. Beghoura, M.A., Boubetra, A., Boukerram, A.: Green software requirements and measurement: random decision forests-based software energy consumption profiling. Requirements Engineering **22**(1), 27–40 (2017)
3. Chauhan, N.S., Saxena, A.: A green software development life cycle for cloud computing. IT Professional **15**(1), 28–34 (2013)
4. Chen, K., Chen, K., Wang, Q., He, Z., Hu, J., He, J.: Short term load forecasting with deep residual networks. IEEE Transactions on Smart Grid (2018)
5. Chen, X.W., Lin, X.: Big data deep learning: challenges and perspectives. IEEE access **2**, 514–525 (2014)
6. Chen, Y., Lin, Z., Zhao, X., Wang, G., Gu, Y.: Deep learning-based classification of hyperspectral data. IEEE Journal of Selected topics in applied earth observations and remote sensing **7**(6), 2094–2107 (2014)
7. Dam, H.K., Tran, T., Pham, T.: A deep language model for software code. arXiv preprint arXiv:1608.02715 (2016)
8. Ding, X., Zhang, Y., Liu, T., Duan, J.: Deep learning for event-driven stock prediction. In: Ijcai. pp. 2327–2333 (2015)
9. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research **11**(Feb), 625–660 (2010)
10. Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: international conference on machine learning. pp. 1050–1059 (2016)
11. Gulshan, V., Peng, L., Coram, M., Stumpe, M.C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al.: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. Jama **316**(22), 2402–2410 (2016)
12. Gupta, R., Pal, S., Kanade, A., Shevade, S.: Deepfix: Fixing common c language errors by deep learning. In: AAAI. pp. 1345–1351 (2017)
13. Hao, S., Li, D., Halfond, W.G., Govindan, R.: Estimating mobile application energy consumption using program analysis. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 92–101. IEEE Press (2013)
14. Kim, D., Hong, J.E., Yoon, I., Lee, S.H.: Code refactoring techniques for reducing energy consumption in embedded computing environment. Cluster Computing pp. 1–17 (2016)

15. Lee, H., Pham, P., Largman, Y., Ng, A.Y.: Unsupervised feature learning for audio classification using convolutional deep belief networks. In: Advances in neural information processing systems. pp. 1096–1104 (2009)
16. Li, D., Halfond, W.G.: An investigation into energy-saving programming practices for android smartphone app development. In: Proceedings of the 3rd International Workshop on Green and Sustainable Software. pp. 46–53. ACM (2014)
17. Li, D., Hao, S., Gui, J., Halfond, W.G.: An empirical study of the energy consumption of android applications. In: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. pp. 121–130. IEEE (2014)
18. Li, D., Lyu, Y., Gui, J., Halfond, W.G.: Automated energy optimization of http requests for mobile applications. In: Proceedings of the 38th international conference on software engineering. pp. 249–260. ACM (2016)
19. Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.Y., et al.: Traffic flow prediction with big data: A deep learning approach. IEEE Trans. Intelligent Transportation Systems **16**(2), 865–873 (2015)
20. Ng, H.W., Nguyen, V.D., Vonikakis, V., Winkler, S.: Deep learning for emotion recognition on small datasets using transfer learning. In: Proceedings of the 2015 ACM on international conference on multimodal interaction. pp. 443–449. ACM (2015)
21. Romansky, S., Borle, N.C., Chowdhury, S., Hindle, A., Greiner, R.: Deep green: modelling time-series of software energy consumption. In: Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on. pp. 273–283. IEEE (2017)
22. Sahin, C., Pollock, L., Clause, J.: How do code refactorings affect energy usage? In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. p. 36. ACM (2014)
23. Shenoy, S.S., Eeratta, R.: Green software development model: An approach towards sustainable software development. In: India Conference (INDICON), 2011 Annual IEEE. pp. 1–6. IEEE (2011)
24. Spencer, M., Eickholt, J., Cheng, J.: A deep learning network approach to ab initio protein secondary structure prediction. IEEE/ACM transactions on computational biology and bioinformatics (TCBB) **12**(1), 103–112 (2015)
25. Wan, M., Jin, Y., Li, D., Gui, J., Mahajan, S., Halfond, W.G.: Detecting display energy hotspots in android apps. Software Testing, Verification and Reliability **27**(6), e1635 (2017)
26. White, M., Tufano, M., Vendome, C., Poshyvanyk, D.: Deep learning code fragments for code clone detection. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. pp. 87–98. ACM (2016)
27. White, M., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D.: Toward deep learning software repositories. In: Proceedings of the 12th Working Conference on Mining Software Repositories. pp. 334–345. IEEE Press (2015)