Journal of Big Data

SHORT REPORT

Open Access

Exploring the efficacy of transfer learning in mining image-based software artifacts

Natalie Best[1], Jordan Ott[2] and Erik J. Linstead[1*]

*Correspondence:
linstead@chapman.edu
[1] Fowler School of Engineering, Chapman University, One University Dr., Orange, CA 92866, USA
Full list of author information is available at the end of the article

Abstract

Background: Transfer learning allows us to train deep architectures requiring a large number of learned parameters, even if the amount of available data is limited, by leveraging existing models previously trained for another task. In previous attempts to classify image-based software artifacts in the absence of big data, it was noted that standard off-the-shelf deep architectures such as VGG could not be utilized due to their large parameter space and therefore had to be replaced by customized architectures with fewer layers. This proves to be challenging to empirical software engineers who would like to make use of existing architectures without the need for customization.

Findings: Here we explore the applicability of transfer learning utilizing models pre-trained on non-software engineering data applied to the problem of classifying software unified modeling language (UML) diagrams. Our experimental results show training reacts positively to transfer learning as related to sample size, even though the pre-trained model was not exposed to training instances from the software domain. We contrast the transferred network with other networks to show its advantage on different sized training sets, which indicates that transfer learning is equally effective to custom deep architectures in respect to classification accuracy when large amounts of training data is not available.

Conclusion: Our findings suggest that transfer learning, even when based on models that do not contain software engineering artifacts, can provide a pathway for using off-the-shelf deep architectures without customization. This provides an alternative to practitioners who want to apply deep learning to image-based classification but do not have the expertise or comfort to define their own network architectures.

Keywords: Deep learning, Transfer learning, UML

Introduction

Despite the recent successes of deep architectures, such as convolutional neural networks, on software engineering data, the lack of sufficiently large training sets for some applications continues to be a substantial hurdle. This requirement has led researchers to label tens of thousands [1] and even millions of images [2] by hand. Recent work has shown that this precludes the use of many off-the-shelf convolutional neural network architectures, requiring empirical software engineering researchers to rely on custom (more compact) architectures [3]. Another possible solution, however, is to leverage transfer learning to deal with large parameter spaces. Through this process models learn in one domain—where data is plentiful—and *transfer* this knowledge to a domain where data is scarce.

One significant limitation in deep learning is data dependence. As computational ability and available algorithms have improved significantly over the years, many deep learning techniques are still held back by the need for massive amounts of labeled truth data. As architectures increase in depth and number of parameters, the amount of data needed to train networks increases as well. When large datasets are not available, or are difficult to curate, researchers must turn to other methods in order to improve their models. Other possible solutions to small amounts of data have been investigated including low shot learning, meta-learning, and data augmentation [3]. Although, even with these other methods to combat small datasets, the bottleneck of large parameter spaces and the computation time needed to train a deep neural network remains. As an example, the very deep convolutional networks developed by the Visual Geometry Group at the University of Oxford, take about 2–3 weeks to fully train the 130–140 million parameters in a network, depending on the architecture [4].

In this paper, we explore transfer learning as a way to combat the issues related to limited data. Many publicly-available, state-of-the-art models already exist and have been trained on huge amounts of data including VGG [4], AlexNet [5], ResNet [6], and Inception [7]. These networks have repeatedly been applied to different tasks from which they were originally trained [1, 8–11]. We will also apply an off-the-shelf architecture, fine tuning it to our task, to show the advantages of knowledge transfer when working with limited data in the software domain. We focus on the classification of unified modeling language (UML) diagrams into class and sequence diagrams from a publicly-available dataset [12]. This dataset has been previously leveraged to demonstrate barriers that arise when applying deep architectures with vast parameter spaces.

The remainder of this paper will be structured as follows. In the next section, we discuss the transfer learning technique, as well as how to apply it and how to assess its results. In our data section, we detail the dataset used in this study and how the images were prepared. The neural network architectures and methods used in our experiments are described in the Methods/Experiments section. We then report the results for each of these architectures at increasing levels of training data. In our final two sections, we discuss various applications of transfer learning and conclude the paper.

Transfer learning

Transfer learning is the process of taking a model trained for one task, where data is more readily available, and applying it to a new but similar task [13]. Traditionally, given two separate tasks, we would have to obtain two distinct training sets and build models for each task. Unfortunately, large amounts of data in every domain are not always available, and in a lot of cases are not always needed if two tasks are similar enough.

When considering how humans learn to do new tasks, they rarely have to start at the absolute beginning—tabula rasa—and typically are building off of similar previous experiences. If one tries to learn a new language, or how to play a new game, one draws on prior knowledge and adapts to complete the task at hand. This is core idea of transfer learning; to learn general features in one domain and apply those features to another, similar domain. In our case, we transfer general features learned when the VGG network has been trained on the ImageNet [14] dataset and fine tune it to the task of UML classification. We choose this classification task for our experiment for three reasons. First, the work in [3] used this same data to demonstrate the inability of deep networks such as VGG-16 to learn features when training samples are limited, requiring custom architectures to be built. Second, UML is sufficiently dissimilar to other objects found in ImageNet that we can be confident that pre-trained models will not have already learned features directly applicable to the classification task. Finally, the automated classification of software artifacts is an essential task when curating data on an Internet-scale as is typically the case in empirical software engineering studies. In our study, we will implement the same networks used by Ott et al. as baselines [3].

When applying transfer learning, a decision must be made to determine how much will be borrowed from the original model. It is common practice to take an established architecture and freeze some amount of the original layers, while fine tuning the rest to the specific needs of a problem. As a result, only the unfrozen layers are trained—resulting in far fewer learnable parameters which decreases the size of the required labeled dataset for training. The amount frozen and fine-tuned is variable depending on the task at hand. We will explore two variations on the VGG-16 architecture, as well as a shallow CNN in this paper. In one VGG network, we fine tune all available weights and see poor accuracy when dealing with small training samples due to the large parameter space that must be learned. In the second, we freeze the majority of weights while fine tuning only the final layer and see accuracy near 90% even at very low numbers of training samples.

In general, when implementing transfer learning, we look in three areas for possible superiority over other networks, as outlined in [15]. First, we may find a higher starting accuracy, at the beginning of training, before the model has been refined further. Second, we could see a steeper or faster rate of improvement of accuracy as training continues. Finally, we look for a higher asymptote, or greater accuracy toward the end of training. In our results, we find that the frozen VGG network exhibits higher accuracy in all three of these areas over the pre-trained VGG and a shallow CNN.

Data

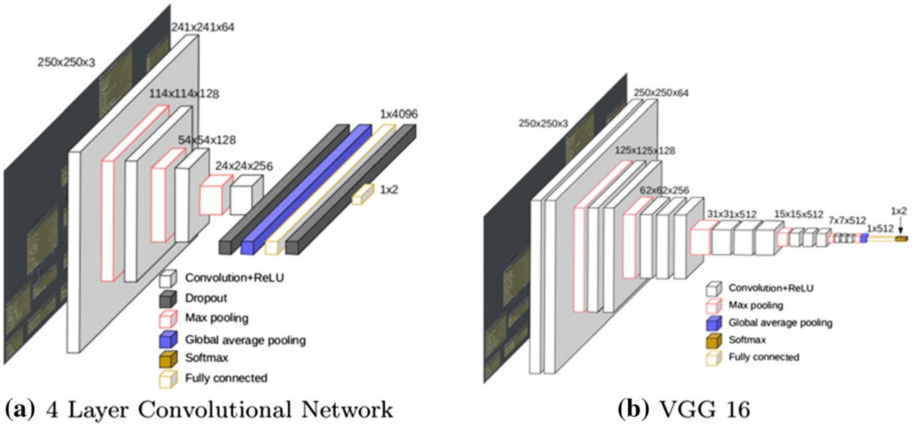
From the Lindholmen Dataset [12], an initial corpus of 14,815 portable network graphics (PNG) images of UML diagrams is obtained. That is then reduced to 13,359 images when only active UML diagrams are considered. Of the active diagrams, there were 11,319 Class Diagrams and 2040 Sequence Diagrams. We resize all images to 250 × 250 pixels for uniformity. To resize a file, we sample the pixels depending on how large the original image was. For example, given a 1024 × 1024 pixel image, every 4th pixel would be used in the x and y direction, or 1024 / 250 = 4. This dataset was chosen for its small size and its relation to software repositories. The VGG-16 networks we include in our tests have been trained on the ImageNet dataset which includes over 1,000,000 natural images belonging to 1000 categories. Although the natural images of ImageNet and UML diagrams exist in quite different domains, we still see improvement in classification when using knowledge transfer.

Methods/experiments

In our experiments, we compare three convolutional neural network (CNN) architectures on their classification ability of UML diagrams. First, we use a simple network with four convolutional layers, max pooling, dropout, and global average pooling layers followed by fully connected dense layers for classification. This network contains 2,260,000 trainable parameters. Two other networks explored are variations of the popular VGG network with sixteen convolutional layers modified to fit the size of our input data [4]. The first VGG we test starts with the original weights and we then allow all 14,715,000 trainable parameters to be updated as we train for our task. Conversely, in the second VGG, we freeze the majority of layers, and then modify and train only the last layer containing only 1026 trainable parameters. The four layer CNN and VGG architectures are shown in Fig. 1. All networks are implemented in Keras with a TensorFlow backend.

These three models were trained as binary classifiers to differentiate UML diagrams as either sequence or class diagrams. To show the advantages of transfer learning, we incrementally increase the available training data in two tests. We begin with 50 samples of each class and increase by increments of 250 to 1800 samples. A second test to show the accuracies at very low samples is performed beginning with 5 samples and increasing by increments of 5 to 50 samples. Upon incrementing the sample size, each network is reset to the same original weights.

Each model was trained for a minimum of 5 epochs and stopped when the accuracy had not improved after a patience of 5 epochs. We implemented fivefold cross validation for robustness. It is common practice to include a patience in order to control training time [16]. Therefore, when a model shows no signs of improving, and we have met an established minimum number of epochs, we are free to stop. For example, in our test of the 1800 diagram sample size, our frozen VGG network quickly reached an accuracy of around 93%, on each fold, after an average of only 15 epochs. Continuing to train would likely not improve our model by any significant amount and could even lead to overfitting.



(a) 4 Layer Convolutional Network

(b) VGG 16

Fig. 1 Visuals of the networks used. **a** The four convolutional layers, interspersed with max pooling for downsampling followed by dropout, max pooling, and fully connected layers for classification. **b** Standard VGG network with sixteen convolutional layers

The code and data to train all models, as well as the learned models themselves, are available publicly at: (removed for anonymity) We hope they, in turn, will be utilized for transfer learning in future deep learning applications on software data.

Results

Figure 2 shows the test accuracy achieved by each network from 50 to 1800 samples of each class, or 100 to 3600 total images respectively. Both the frozen VGG and 4 layer CNN are eventually able to classify the given diagrams with about 90% accuracy given a sufficient amount of samples. Although, we see a significant difference in the starting accuracies as well as faster convergence.

However, we are also interested in the best accuracy achievable with the least amount of data. The frozen VGG is able to classify with an about 80% accuracy after only 100 total training samples while the 4 layer CNN falls short at about 52% accuracy. As can be expected, the VGG that was left free to train the massive number of parameters within its network, also performs poorly, barely reaching 50% accuracy. In which case, it would be no better than simply flipping a fair coin to classify each diagram. The tiny amount of training data given to this network is, of course, nowhere near enough to train all 14 million parameters.

Figure 3 shows the training accuracy for all three networks when given 5 to 50 samples of each class, or 10 to 100 total images. We include this figure to demonstrate the superiority of the frozen VGG over both networks especially at very low samples. Even with only 10 total samples, the frozen VGG is able to classify the UML diagrams with an average 73% accuracy, compared to an accuracy of only 50% for both other networks.

Class activation mapping allows us to investigate further what parts of an image a convolutional network uses to make its prediction, as well as ensure those features make sense [17]. Using the Keras Visualization Toolkit [18], we produced CAM results for one UML sequence diagram and one class diagram. CAM results are shown in Figs. 4 and 5 for the frozen VGG-16 network trained on 1800 sample images from each class. CAM

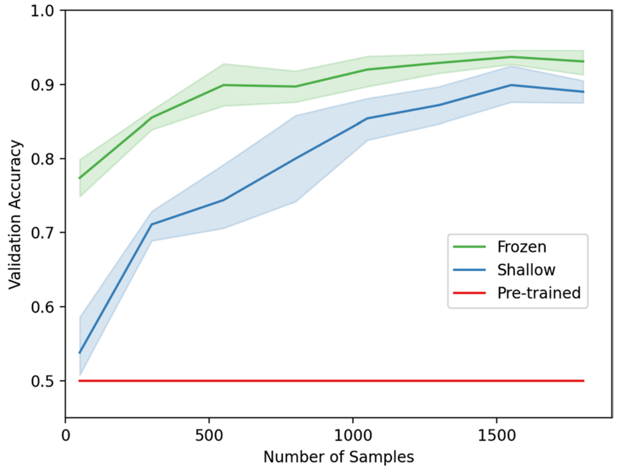


Fig. 2 Displayed above is the number of training samples from each category, from 50 to 1800, vs the validation set accuracy achieved by each model. For robustness, 5 trials were run for all training samples tested. The color bands indicate the distribution of results from the 5 trials

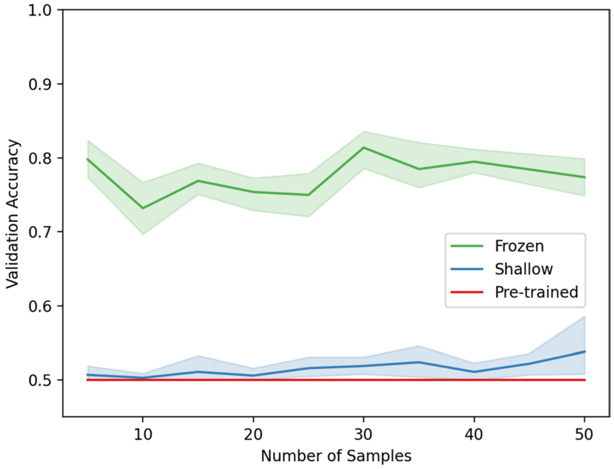


Fig. 3 Shown above is the accuracy achieved by each network at the corresponding sample sizes, from 5 to 50 samples in each UML category. For robustness, 5 trials were run for all training samples tested. The color bands indicate the distribution of results from the 5 trials

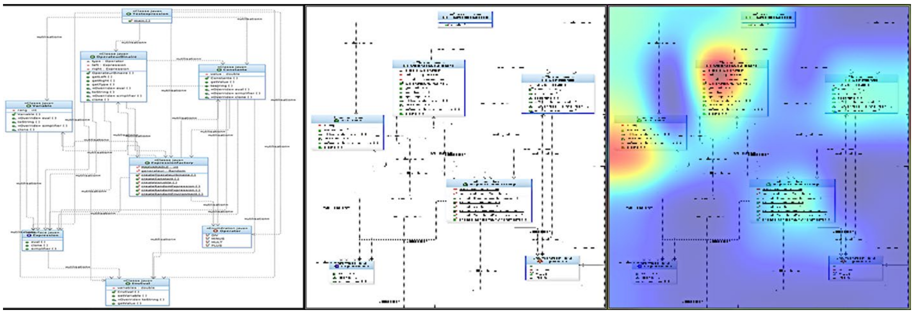


Fig. 4 Class activation mapping prediction for a selected UML class diagram, original image on the left, resized image in the middle, heatmap indicating significant features on the right

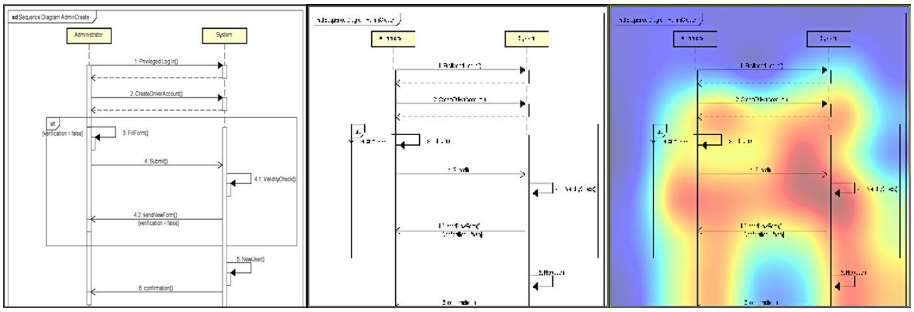


Fig. 5 Class activation mapping prediction for a selected UML sequence diagram, original image on the left, resized image in the middle, heatmap indicating significant features on the right

produces a heat map highlighting the regions most heavily weighted by the network. We are able to see clearly that the network learns features specific to sequence and class diagrams. Specifically, in class diagrams, the boxes containing class attributes and methods have been highlighted. Conversely, in sequence diagrams, the vertical lifelines are more significant.

We also compared the computational cost of training only the last layer of the frozen network to the entire unfrozen network. Training time for each model varies based on the number of epochs completed but generally, each one of these models can be fully trained in 30 minutes. The VGG model with frozen weights averages a little less than half a second faster, per epoch, than the VGG model training all layers. The difference results from less computations required during the backpropagation of errors in models with frozen weights. As the dataset increases in size one can expect the difference in time between the two models to increase as more batches are completed per epoch. We can also compare our computation time to the computation time needed to train the original VGG-16. No doubt the difference in dataset size has an effect in reducing computation time, as the original network was trained on the large ImageNet dataset, but so would the number of trainable parameters. Simonyan and Zisserman, the creators of the VGG network, report that training a single network took 2–3 weeks depending on the specific architecture [4].

Discussion

The classification of UML diagrams has been studied through a variety of machine learning techniques. Ho-Quang et al. [19] proposed a logistic regression model using 19 of their 23 proposed features for classifying UML and non-UML class diagrams (CD). When trained on a corpus of 1300 images, their model achieved 96% accuracy for UML-CD and 91% of accuracy for non-UML CD. Years later, Ho-Quang et al. [19] furthered their work to differentiate between diagrams that were hand-made as part of the forward-looking development process (FwCD), and diagrams that were reverse engineered from the source code (RECD). However instead of classifying the images directly, the authors extract various features and implement a random forest model to achieve 90% accuracy in distinguishing the two types of class diagrams. In another study, using a corpus of 1300 UML and non-UML images, Hjaltason et al. [20] trained a support vector machine (SVM) with an average classification accuracy of 92.05% . Moreno et al. [21] conducted a similar study to classify web images as UML and non-UML class diagrams using a rule based approach. By extracting features from the images, in a corpus of 19000 web images, their algorithm reached an accuracy of 95%.

While we believe this is one of the first attempts to study the applicability of transfer learning to images within software engineering, transfer learning in general has been studied in many domains and aided in the development of powerful machine learning models. Authors in [22], propose the use of 'bellwethers', or the software project whose data yields the best predictions on all other projects. They argue that a simple transfer learner constructed from the bellwhether's data should be used as a baseline for future transfer learning work. In their study, they found that the simple transfer learner yielded comparable predictions to other more complex models. Effort estimation is just one area within the software domain where transfer learning has proven valuable. In an extension of previous work, Kocaguneli et al. [23], explore transfer learning in the field of effort estimation and for both the cross-company learning problem and cross time learning problem. Similarly, Ying et al. [24] also investigate transfer learning for cross-company defect prediction in software. Another study, takes one step further to include canonical correlation analysis into their study of cross-company defect prediction [25]. In physical applications, such as robotics, training samples can be especially costly, both in time and energy costs. In order to learn most efficiently while balancing these costs, transfer learning has been employed to predict the performance of physical systems under different configurations [26]. As a result, models do not need to be trained from scratch for each time and existing configurations can be adapted with few additional training examples.

Shin et al. [8] investigated the effectiveness of CNN architectures and transfer learning in detecting thoraco-abdominal lymph nodes and classifying interstitial lung disease from images. The authors achieve state-of-the art performance and find transfer learning to be beneficial despite the natural images used to train ImageNet being significantly different from medical images. Another study applied transfer learning to four medical imaging applications in 3 specialties including radiology, cardiology, and gastroenterology [27]. Their experiments transferred weights from ImageNet layer-wise, using none, a few, or many layers and found that transferring a few layers improved performance compared to training from scratch.

As stated previously, transfer learning in the space of software imagery was motivated by the work in [3]. Here the authors showed definitively that deep networks like VGG were unable to compete with smaller architectures when labeled data was sparse. A viable workaround was to create custom, shallower architectures that were compatible with available data volumes. The work presented here shows that off-the-shelf architectures can be used, but demand more efficient learning solutions—specifically the kinds produced via transfer learning.

The ultimate goal of the work in this paper is to make deep learning and off-the-shelf convolutional architectures more available to empirical software engineering researchers who have a need to classify software artifacts. While large, labeled datasets are readily available for textual source code, for image-based artifacts such as UML, the curation of large volumes of training data continues to be a hurdle. This complicates the use of standard deep architectures such as VGG. The results achieved here indicate that transfer learning provides a path forward to researchers who wish to apply deep learning architectures to software artifact classification when only modest amounts of data are available. Specifically, pre-training with ImageNet using standard VGG architectures results in excellent classification performance of class and sequence diagrams despite the fact that the ImageNet dataset itself contains no examples of these artifacts. These benefits are in addition to those provided by transfer learning when massive training sets are available, in particular shorter model training times.

As with all work, there are some limitations to the experimental results presented here that are worth noting. First of all, experiments make use of only one data set based on UML. In future work, we will apply our transfer learning approach to other image-based software artifacts. Secondly, the classification task detailed here is binary, and discriminates only between class and sequence diagrams. It will be important to generalize this work to multi-class classification problems where only small amounts of training data are available. Finally, it would also be useful to assess the performance of datasets other than ImageNet as a basis for transfer learning.

Conclusion

Transfer learning allows us to take, in effect, a shortcut in training deep architectures. In this paper, we extended previous work regarding the application of machine learning techniques for classification of UML images. Given limited data, it is nearly impossible to train a network with the depth and substantial number of parameters as in VGG. However, by transferring knowledge learned from one task to another, we are able to tune off-the-shelf deep architectures and achieve high classification accuracy, rather than having to design new architectures with fewer layers and smaller parameter spaces to learn. Most importantly, the knowledge that forms the basis of the transfer learning needs no previous exposure to artifacts from the software domain, suggesting that transfer learning can be applied broadly to applications of deep learning within empirical software engineering.

Our experimental results have show training is positively effected by transfer learning even when the number of samples shown to the network is kept small. In contrast, even a smaller network with substantially fewer parameters is unable to learn as well. As a control, we also tested an off-the-shelf VGG and allowed the entire architecture containing over 14 million parameters to train. As expected, this network failed to improve beyond 50% accuracy even when shown the maximum number of samples tested.

In addition to affirming the utility of utilizing transfer learning for mining software artifacts, our results suggest that as a research community we should be more proactive in curating machine learning models trained on software data, in addition to the software data itself. Such repositories of pre-trained models would allow empirical software engineering researchers to apply transfer learning to new applications using models already tuned using software data of various types.

Abbreviations

CAM: Class activation mapping; CNN: Convolutional neural network; GPU: Graphics processing unit; PNG: Portable network graphics; SVM: Support vector machine; UML: Unified modeling language; VGG: Visual geometry group.

Acknowledgements

The authors wish to thank Experian Consumer Services for their support of the Machine Learning and Assistive Technology Lab at Chapman University.

Authors' contributions

NB and JO implemented the deep learning code, carried out experiments, and contributed to the manuscript. NB gathered and cleaned data for machine learning and contributed to the manuscript. EJL designed the experiments, contributed to the manuscript, and provided computing resources. All authors read and approved the final manuscript.

Funding

The authors declare that no funding was received for this work.

Availability of data and materials

All code required to train and run the CNNs described above is available by written request to the corresponding author.

Competing interests

The authors declare that they have no competing interests.

Author details

[1] Fowler School of Engineering, Chapman University, One University Dr., Orange, CA 92866, USA. [2] School of Information and Computer Science, University of California, Irvine, Irvine, CA, USA.

Received: 7 March 2020 Accepted: 30 July 2020
Published online: 08 August 2020

References

1. Ott J, Atchison A, Harnack P, Bergh A, Linstead E. A deep learning approach to identifying source code in images and video. 2018. p. 376–86. https://doi.org/10.1145/3196398.3196402.

2. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, et al. Imagenet large scale visual recognition challenge. Int J Comput Vis. 2015;115(3):211–52.
3. Ott J, Atchison A, Linstead EJ. Exploring the applicability of low-shot learning in mining software repositories. J Big Data. 2019;6(1):35. https://doi.org/10.1186/s40537-019-0198-z.
4. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition; 2014. arXiv:1409.1556.
5. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems - Volume 1. NIPS'12, pp. 1097–1105. USA: Curran Associates Inc.; 2012. http://dl.acm.org/citation.cfm?id=2999134.2999257.
6. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. 2015. CoRR arXiv:1512.03385.
7. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: Computer vision and pattern recognition (CVPR). 2015. arXiv:1409.4842.
8. Shin H, Roth HR, Gao M, Lu L, Xu Z, Nogues I, Yao J, Mollura D, Summers RM. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Trans Med Imaging. 2016;35(5):1285–98. https://doi.org/10.1109/TMI.2016.2528162.
9. Bayramoglu N, Heikkilä J. Transfer learning for cell nuclei classification in histopathology images. In: Hua G, Jégou H, editors. Computer vision—ECCV 2016 workshops. Cham: Springer; 2016. p. 532–9.
10. Ott J, Atchison A, Harnack P, Best N, Anderson H, Firmani C, Linstead E. Learning lexical features of programming languages from imagery using convolutional neural networks. In: Proceedings of the 26th conference on program comprehension. New York: ACM; 2018. p. 336–9.
11. Alahmadi M, Hassel J, Parajuli B, Haiduc S, Kumar P. Accurately predicting the location of code fragments in programming video tutorials using deep learning. In: Proceedings of the 14th international conference on predictive models and data analytics in software engineering. New York: ACM; 2018. p. 2–11.
12. Hebig R, Quang TH, Chaudron MRV, Robles G, Fernandez MA. The quest for open source projects that use uml: Mining github. In: Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems. MODELS '16. New York: ACM; 2016. p. 173–83. https://doi.org/10.1145/2976767.2976778.
13. Pan SJ, Yang Q. A survey on transfer learning. IEEE Trans Knowl Data Eng. 2010;22(10):1345–59. https://doi.org/10.1109/TKDE.2009.191.
14. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. ImageNet: a large-scale hierarchical image database. In: CVPR09; 2009.
15. Olivas ES, Guerrero JDM, Sober MM, Benedito JRM, Lopez AJS. Handbook of research on machine learning applications and trends: algorithms, methods and techniques—2 volumes. Hershey: Information Science Reference - Imprint of: IGI Publishing; 2009.
16. Bengio Y. Practical recommendations for gradient-based training of deep architectures. 2012. CoRR arXiv:1206.5533.
17. Zhou B, Khosla A, Lapedriza À, Oliva A, Torralba A. Learning deep features for discriminative localization. 2015. CoRR arXiv:1512.04150.
18. Kotikalapudi R. contributors: keras-vis. GitHub; 2017.
19. Ho-Quang T, Chaudron MRV, Samuelsson I, Hjaltason J, Karasneh B, Osman H. Automatic classification of UML class diagrams from images. 2014 21st Asia-Pac Softw Eng Conf. 2014;1:399–406.
20. Hjaltason J, Samuelsson I. Automatic classification of uml class diagrams through image feature extraction and machine learning. 2015.
21. Moreno V, Génova G, Alejandres M, Fraga A. Automatic classification of web images as UML diagrams. In: Proceedings of the 4th Spanish conference on information retrieval. CERI '16. New York: ACM; 2016. p. 17–1178. https://doi.org/10.1145/2934732.2934739.
22. Krishna R, Menzies T. Bellwethers: a baseline method for transfer learning. 2017. arXiv e-prints. arXiv:1703–06218.
23. Kocaguneli E, Menzies T, Mendes E. Transfer learning in effort estimation. Empirical Softw Eng. 2015;20(3):813–43. https://doi.org/10.1007/s10664-014-9300-5.
24. Ma Y, Luo G, Zeng X, Chen A. Transfer learning for cross-company software defect prediction. Inf Softw Technol. 2012;54(3):248–56. https://doi.org/10.1016/j.infsof.2011.09.007.
25. Jing X, Wu F, Xiwei D, qi F, Xu B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. 2015. p. 496–507. https://doi.org/10.1145/2786805.2786813.
26. Jamshidi P, Velez M, Kästner C, Siegmund N, Kawthekar P. Transfer learning for improving model predictions in highly configurable software. 2017. CoRR arXiv:1704.00234.
27. Tajbakhsh N, Shin J, Gurudu S, Hurst R, Kendall C, Gotway M, Liang J. Convolutional neural networks for medical image analysis: full training or fine tuning? IEEE Trans Med Imaging. 2016;35(5):1299–312. https://doi.org/10.1109/TMI.2016.2535302.

Publisher's Note