Ott *et al. J Big Data (2019) 6:35*
https://doi.org/10.1186/s40537-019-0198-z

Journal of Big Data

SHORT REPORT Open Access

Exploring the applicability of low-shot learning in mining software repositories

Jordan Ott[1], Abigail Atchison[2] and Erik J. Linstead[2*]

*Correspondence:
linstead@chapman.edu
[2] 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: Despite the well-documented and numerous recent successes of deep learning, the application of standard deep architectures to many classification problems within empirical software engineering remains problematic due to the large volumes of labeled data required for training. Here we make the argument that, for some problems, this hurdle can be overcome by taking advantage of low-shot learning in combination with simpler deep architectures that reduce the total number of parameters that need to be learned.

Findings: We apply low-shot learning to the task of classifying UML class and sequence diagrams from Github, and demonstrate that surprisingly good performance can be achieved by using only tens or hundreds of examples for each category when paired with an appropriate architecture. Using a large, off-the-shelf architecture, on the other hand, doesn't perform beyond random guessing even when trained on thousands of samples.

Conclusion: Our findings suggest that identifying problems within empirical software engineering that lend themselves to low-shot learning could accelerate the adoption of deep learning algorithms within the empirical software engineering community.

Keywords: Deep learning, Low-shot learning, UML

Introduction

In the past couple of years, applications of deep learning to mining software repositories have grown in number and diversity of methods [1–5]. Fueled in part by easy-to-use libraries and graphics processing unit (GPU) computing, deep architectures have facilitated new avenues for research, often producing results that far surpass previous techniques. However, despite their advantages, the huge amount of labeled truth data traditionally required to train deep architectures for classification tasks, as well as the computational time required to iteratively improve such models, remains a substantial bottleneck [6]. As a result, some researchers are forced to turn away from deep architectures, despite the fact that for certain tasks (like image analysis and computer vision), deep learning consistently outperforms alternative algorithms and methodologies.

Low-shot learning refers to the practice of training machine learning models, including deep neural networks, using far fewer samples of each classification category than what is typically standard practice. In the extreme case, training data consists of only one instance for each target class, which is known as one-shot learning [7]. These approaches are useful in situations when data is scarce or costly to acquire, but discriminative, generalizable features can still be learned from a limited training set due to the properties of the data itself. To avoid overfitting, however, models may have to be appropriately modified to reduce the total number of parameters learned from data. Low-shot learning has been receiving increasing interest from the machine learning research community, especially for the task of image classification. Despite its promise, however, it has to date been unexplored in the context of mining software, and most work in this area leveraging deep learning has relied on the curation of large training sets with thousands of training examples.

This paper provides a proof-of-concept for the application of low-shot learning to mining software artifacts. In particular, we focus on the task of classifying unified modeling language (UML) diagrams from a recently-published, publicly-available dataset [8]. To do this, we leverage convolutional neural networks (CNNs) to discriminate between class diagrams and sequence diagrams. As a baseline, we use an off-the-shelf VGG architecture. We find that even with a training set of 1800 instances it fails to outperform random guessing due to the vast parameter space it must learn. We then show that using low-shot learning with a custom architecture we are able to perform substantially better than random guessing (approximately 70%) with only 100 training instances. This performance increases to almost 90% accuracy with a few hundred instances. . .one or two orders of magnitude fewer than what would be required to achieve similar performance with VGG. However, this also requires appropriately modifying the number of layers in the network to reduce the total parameter space.

While our proof-of-concept focuses on the classification of UML diagrams, the primary contribution of this paper is to demonstrate the efficacy of low-shot learning to the empirical software engineering domain in general. Traditionally, the majority of software engineering artifacts (code, requirements, etc.) have been represented natively as text, thus lending themselves to machine learning algorithms grounded in text mining and natural language processing. However, with the popularity of online code tutorials and public repositories such as Github, software data is increasingly encoded in imagery and video formats, necessitating the application of computer vision algorithms. Compared to general image training datasets like Imagenet, however, the amount of available training data is substantially smaller. Low-shot learning represents an attractive option for researchers in this domain that wish to leverage techniques like CNNs but do not have at their disposal enormous training sets. These results presented here indicate that low-shot learning is a viable avenue for integrating deep architectures into empirical software engineering research while avoiding some of the burdens that often come along with them.

Deep architectures and profuse parameters

Deep architectures refer to any artificial neural network that consists of more than one hidden layer. While this can be achieved using typical fully-connected, feed-forward networks with sigmoidal activation (eg. logistic) functions, most contemporary deep learning efforts are focused on some variant of recurrent or convolutional architectures due to their ability to model input data with temporal or spatial relationships, respectively. In this paper we focus on the latter, as our goal is to leverage deep learning to classify software artifacts consisting of UML class and sequence diagram images. CNNs represent the current state-of-the-art in image classification, and have been used successfully in domains ranging from medicine to remote sensing.

Structured input data such as images lose their spatial relationships when passed through traditional fully-connected architectures. This is problematic for applications such as computer vision where the spatial relationship of pixels conveys critical information. Convolutional neural networks maintain spatial relations between pixels by convolving the input space with a multidimensional weight matrix (in contrast to the matrix multiplication used in fully-connected artificial neural networks), commonly referred to as a filter. Filters represent patterns to be detected in the input image, and are learned during training using the same gradient descent procedure (backpropagation) employed for feed-forward architectures.

CNNs use a shared weight paradigm to reduce the number of trained parameters, and as a result scale better compared to their fully-connected counterparts. Weight-sharing in CNNs is typically associated with two primary functions. The first is to reduce the number of free parameters that need to be stored or updated during learning. This can be important in applications where storage space or training data is limited, or where overfitting is a danger. However, despite weight sharing, the number of parameters that need to be learned grows quickly with every added layer. The second function of weight sharing is to apply the exact same operation at different locations in the input data to process the data uniformly and provide a basis for invariance, typically translation invariant recognition in computer vision applications. This mitigates the need to manually translate input images as part of training.

A driving factor in the adoption of convolutional networks across research domains is the availability of off-the-shelf implementations that have been widely studied by the deep learning research community. Architectures such as VGG (138 million parameters) [9], AlexNet (60 million parameters) [10], ResNet (25 million parameters) [11], and Inception (23 million parameters) [12] have yielded results that far outperform other non-deep learning methodologies on benchmark datasets such as MNIST [13] and CIFAR-10 [14]. Implementations in high-level languages such as Python are easily downloadable on the web, and within minutes researchers can be training these networks on their own data. However, these networks are all very deep, require millions of parameters to be learned, and rely on thousands or tens-of-thousands of training instances. Learning such large numbers of parameters is feasible for datasets such as MNIST and CIFAR-10, which provide 60,000 and 50,000 labeled training images, respectively. Curating training sets of such sizes for open research questions in empirical software engineering, however, can provide a substantial barrier, due to both the time and cost to gather and label the data. Thus, there is a clear motivation to identify research problems in our community where low-shot learning is feasible, as well as explore deep learning architectures that are capable of supporting it.

An application of low-shot learning

As a proof-of-concept for low-shot learning in the software mining domain, we decided on the problem of automatic classification of UML diagrams. We found this to be a compelling application for several reasons. First of all, UML design artifacts are prevalent in open source software repositories but have received relatively little attention from our community. Secondly, researchers have currently made available a large collection of labeled UML diagrams [8], thus facilitating other research groups to reproduce and extend the work presented here. Finally, we believe that classifying sequence and class diagrams is a natural binary classification task for low-shot learning and each type diagram has tell-tale features that should be learnable with a relatively few number of instances and also generalizable to unseen data. This rationale is of course grounded in our own experiences with human learning. A young child does not need to see thousands of instances of dogs and cats in order to learn to differentiate between them. Similarly, class and sequence diagrams are different enough that the same should extend to artificial neural networks. It is important to emphasize that the goal of this paper is not to achieve near-perfect classification results, or even attempt to surpass performance of other machine learning techniques applied to the same classification problem. Rather, we focus only on the feasibility of low-shot learning for this particular classification problem.

Data

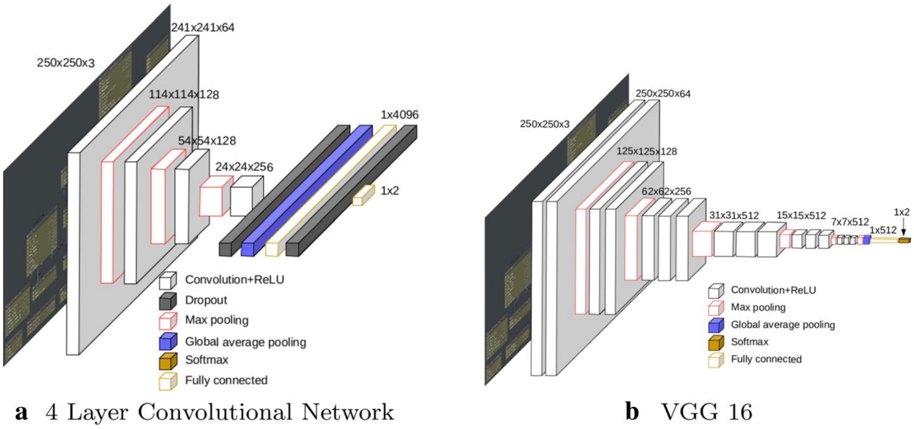
UML Diagrams were extracted from The Lindholmen Dataset [15]. For the purposes of this model we extracted only portable network graphics (PNG) images with a specific UML Diagram class from the database resulting in an initial UML corpus of 14,815 potential, unique UML files. After the extraction of all active urls the final corpus consisted of 13,359 PNG images. In this total there were 11,319 Class Diagrams and 2040 Sequence Diagrams. For uniformity, all images were resized to 250×250.

Methods

Two convolutional architectures were employed in this study. A smaller network with four convolutional layers (Fig. 1a) for low-shot learning and a VGG network with sixteen convolutional layers [9] was used as a baseline. Both networks were implemented in Keras with a TensorFlow backend.

The smaller network was composed of four convolutional layers, interspersed with max pooling layers, followed by dropout (for regularization), global average pooling, and fully connected layers for classification. The network contains a total of 2,260,000 trainable parameters. The VGG network shown in Fig. 1b is a very popular model used across many domains, and was chosen because it has recently been applied to software mining [2]. The VGG model has a convenient architecture in which multiple convolutional operations occur in succession, followed by a max pooling layer for down-sampling. Following the convolutional layers are fully connected layers for classification. The number of neurons in fully connected layers was changed to 512 to account for the size of the input volume. This modification gives our VGG model 14,715,000 (opposed to the original 138 million) trainable parameters. Nearly seven times the number of parameters learned by our smaller network.

Both networks were trained to classify UML diagrams. To test the validity of low-shot learning the amount of training data was varied, starting from fifty training samples of each category (100 total) up to 1800 with increments of 250. For robustness, 10 trials were run for every increment, i.e. the number of training samples of each category. Each



a 4 Layer Convolutional Network **b** VGG 16

Fig. 1 Convolutional networks used for low-shot learning. **a** The four convolutional layers, interspesed with max pooling for downsampling followed by dropout, max pooling, and fully connected layers for classification. **b** Standard VGG network with sixteen convolutional layers

trial randomly selected a new training set and a held out test set of 200 samples, 100 from each category. Training and testing was carried out on a 32-core i7 server-class machine with 512 GB physical memory and 2 NVidia P100 GPUs. The total time to run all experiments across different sample sizes was approximately 5 h.

Results

Figure 2 displays the test accuracy achieved by the respective networks after training on the corresponding number of samples from each category. The results indicate that the shallow network performs well (substantially above random guessing) after seeing only fifty samples from each category, achieving a test accuracy of 70%. The accuracy continues to rise as more training samples are added, jumping to about 80% after 300 samples, and reaching 90% accuracy after 750 samples. Performance levels off around 92%, after 1000 samples are seen. Conversely, the VGG network never scores better than random

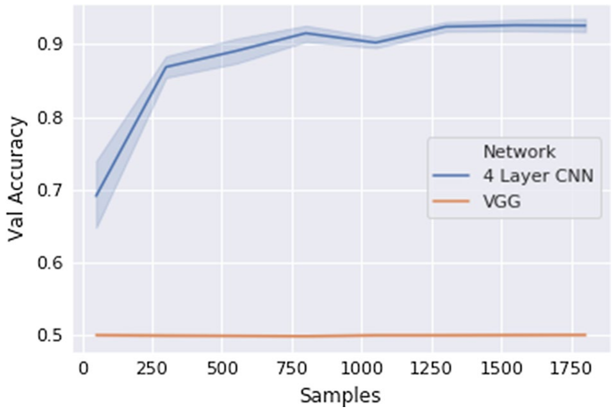


Fig. 2 Displayed above is the number of training samples from each category vs the validation set accuracy achieved by the corresponding model. For robustness, 10 trials were run for every amount of training samples tested. The color bands indicate the distribution of results from the 10 trials

guessing with 50% on testing data, even when trained on 3600 images. Thus, even with 36 times the amount of training data, the huge parameter space of the standard VGG network prohibits it from learning any generalizable features, resulting in accuracy 20% less than our smaller architecture trained with only 100 instances.

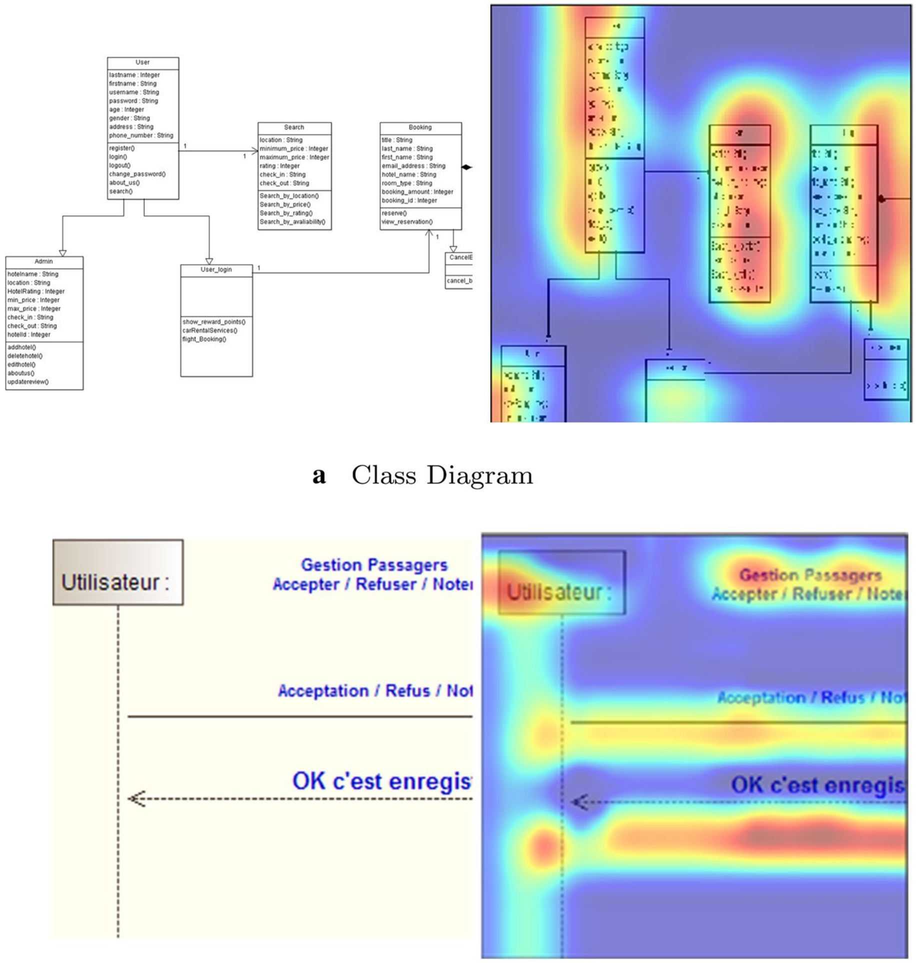
Class activation mapping (CAM) gives convolutional networks tremendous localization ability despite being trained on image-level labels [16]. The Keras Visualization Toolkit [17] is used to produce CAM results. Using CAM, we are able to visualize what regions of input images the network attends to when making its classification prediction. This allows us to ensure the network is learning features directly related to the type of UML diagram.

CAM results are shown in Fig. 3 for the 4 convolutional layer architecture trained with only 100 instances. The heatmap produced can be interpreted by the degree of redness in a given region. The more red a region is, the more weight the network associates with features in that area to formulate its output prediction. We see that the network, despite the small amount of training data, has clearly learned discernible differences in class and sequence diagrams. For example, in the sequence diagram the lifelines and messages have been learned as key features for classification. Similarly, in the class diagram, the boxes containing class attributes and methods have been learned as differentiators, while the lines representing dependencies have been ignored.

Related works

While we believe this paper is the first to apply deep learning to the task, the automatic classification of UML diagrams is not novel. Ho-Quang et al. presented a Logistic Regression model to identify UML from non-UML images with an accuracy of 91.4%. This model was trained on a corpus of 1300 images [18]. In 2015 Hjaltason et al. [19] utilized support vector machines (SVMs) trained on a corpus of 1300 UML and non-UML images, producing an average classification accuracy of 92.05%. Moreno et al. addressed the high processing times in these previous approaches by proposing a rule based approach [20]. These rules were extracted from a corpus of nearly 19,000 web images (UML and non-UML), and used a training set of 715 images to identify UML images with a 95% accuracy. Though related to classifying UML, these studies focus only classifying UML from non-UML, and not differentiating UML diagram types. Thus, we cannot directly compare previous performance numbers to those reported in this paper using low-shot learning. Given that discriminating UML from non-UML is a simpler learning problem than classifying among UML types, we believe that the low-shot paradigm would provide competitive results.

Low-shot learning has the established potential to remedy multiple problems within the machine learning community. As demonstrated in this study, applying low-shot learning to classification problems can produce promising accuracy metrics on a fraction of the training data. This method serves to benefit more domains than empirical software engineering. In [21] Yang et al. applied one shot learning to gesture data [22, 23] with a classification accuracy of 80%. Fei-Fei et al. demonstrated a Bayesian approach object classification with a recognition rate as high as 82% given only one training example of each class [24]. Wolf et al. used low-shot learning for both insect classification



a Class Diagram

b Sequence Diagram

Fig. 3 Original image left, CAM prediction on down-sampled UML diagrams. **a** A class diagram and **b** A sequence diagram. The heatmap of the class activation mapping identifies significant features for classification

and facial identification. The optimal model presented in their research performed with 87% accuracy rate for low-shot insect classification, however the same model achieved an accuracy rate of 42% for low-shot facial classification [25].

This range in success in low-shot learning even when applying the same model demonstrates that while the practice of low-shot learning has yielded strong results in some domains, including the one found within this study, there are indeed research areas where low-shot classification accuracy lags unjustifiably behind that of models which use a larger corpus of training data. This is further demonstrated in Lake et al. application of one-shot learning to character recognition [26] which reached an average accuracy of 54.9%, falling short of the 98.2% accuracy in identifying handwritten digits achieved by Nair et al. [27].

In cases such as these low-shot learning for classification may not yet, or ever, be a realistic alternative. Nonetheless, low-shot learning can aid in the initial labeling of datasets as shown by Xu et al. whose study presented a low-shot learning model capable of expanding the "vocabulary" of an animal species dataset to include new species [28]. When applying deep architectures to classification problems obtaining large amounts of labeled data has served as a first hurdle researchers must overcome, providing an alternative to this labor-intensive labeling step through low-shot learning helps to relieve this burden placed on researchers, potentially allowing for a wider application of deep architectures.

Conclusion and future work

Here we have explored, for the first time, the application of low-shot learning to mining software artifacts. Our results indicate that if the right classification problem is thoughtfully paired with an appropriate deep architecture, it is possible to achieve reasonable results with substantially smaller training sets than what are typically leveraged for deep learning. This is not meant to suggest, of course, that deep learning is the right machine learning tool for every job. However, for tasks where deep learning is known to outperform alternatives, leveraging a low-shot approach can allow for the use of these algorithms when small training sets would otherwise prohibit them. There is still much work to be done in this vein, but we hope this paper, and our initial results, will serve as a call to action for the software mining community to identify and explore other tasks that lend themselves to low-shot learning.

As with all work, the results presented here have their limitations. Our experimental results consider only a binary image classification problem for UML, when there are many more UML diagram types than just class and sequence diagrams. A useful extension of this work would be to extend to other diagram types beyond the two used here. Similarly, previous applications of deep learning have focused on source code as image data, and identify such images using traditional learning with convolutional neural networks. Future work should apply low-shot learning to these problems, as well as other problems, for example the automatic classification of design patterns from design artifacts. These are only a few example, however. In theory our low-shot approach can be applied to any software mining problem in which the data is encoded as imagery instead of traditional textual encodings.

In our experiments, both the standard VGG network and smaller CNN were trained from random initialization points. Recently, the machine learning community has been exploring pre-training and transfer learning as mechanisms for improving the accuracy of deep architectures as well as reducing the amount of time and training data required. We are currently expanding the work presented here to include experiments with both transfer learning and pre-training. These techniques may make it possible for a standard deep architecture, like VGG, to achieve better than random guessing using a low-shot paradigm.

Ultimately, there are many machine learning tasks within the software mining community for which low-shot learning will never be feasible due to their intrinsic complexity. However, by making a concerted effort to tease out problems where it is applicable, we stand to reap the benefit of bleeding edge algorithms without paying the up-front costs associated with collecting and labeling huge training corpora. For a community so focused on thinking about data on an Internet scale, the real challenge will be reminding ourselves that sometimes less is more.

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.

Authors' contributions

JO implemented the deep learning code, carried out experiments, and contributed to the manuscript. AA 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.

Author details

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

Acknowledgements

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

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

All code required to train and run the CNNs described above is publicly available here: https://github.com/jordanott/OneShot

Funding

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

Publisher's Note

Received: 6 March 2019 Accepted: 26 April 2019
Published online: 06 May 2019

References

1. Tufano M, Watson C, Bavota G, Di Penta M, White M, Poshyvanyk D. Deep learning similarities from different representations of source code. In: Proceedings of the 15th international conference on mining software repositories. MSR '18. New York: ACM; 2018, p. 542–53.
2. Ott J, Atchison A, Harnack P, Bergh A, Linstead E. A deep learning approach to identifying source code in images and video. In: Proceedings of the 15th international conference on mining software repositories. MSR '18. New York: ACM; 2018, p. 376–86.
3. 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. ICPC '18. New York: ACM; 2018, pp. 336–9.
4. 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. PROMISE'18. New York: ACM; 2018 p. 2–11.
5. Ponzanelli L, Bavota G, Mocci A, Oliveto R, Di Penta M, Haiduc SC, Russo B, Lanza M. Automatic identification and classification of software development video tutorial fragments. IEEE Trans Softw Eng. 2018;14:1.
6. Fu W, Menzies T. Easy over hard: a case study on deep learning. In: Proceedings of the 2017 11th joint meeting on foundations of software engineering. ESEC/FSE 2017. New York: ACM; 2017, p. 49–60.
7. Fei-Fei L, Fergus R, Perona P. One-shot learning of object categories. IEEE Trans Pattern Anal Mach Intell. 2006;28(4):594–611.
8. Robles G, Ho-Quang T, Hebig R, Chaudron M R V, Fernandez M A. An extensive dataset of uml models in github. In: Proceedings of the 14th international conference on mining software repositories. MSR '17. Piscataway: IEEE Press; 2017, p. 519–22.
9. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014. arXiv preprint arXiv:1409.1556
10. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th international conference on neural information processing systems, vol 1. NIPS'12. Curran Associates Inc. 2012, p. 1097–105. http://dl.acm.org/citation.cfm?id=2999134.2999257
11. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 770–8.
12. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, p. 1–9.
13. LeCun Y. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

14. Krizhevsky A. Learning multiple layers of features from tiny images. Citeseer: Technical report; 2009.
15. Hebig R, Quang T H, Chaudron M R V, 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.
16. Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A. Learning deep features for discriminative localization. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 2921–9.
17. Kotikalapudi R. contributors: keras-vis. GitHub. 2017
18. Ho-Quang T, Chaudron M R, Samúelsson I, Hjaltason J, Karasneh B, Osman H. Automatic classification of UML class diagrams from images. In: Software engineering conference (APSEC), 2014 21st Asia-Pacific, vol. 1. IEEE; 2014, p. 399–4060.
19. Hjaltason J, Samúelsson I. Automatic classification of UML class diagrams through image feature extraction and machine learning. 2015.
20. 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. ACM; 2016, p. 17.
21. Yang Y, Saleemi I, Shah M. Discovering motion primitives for unsupervised grouping and one-shot learning of human actions, gestures, and expressions. IEEE Trans Pattern Anal Mach Intell. 2013;35(7):1635–48.
22. Lin Z, Jiang Z, Davis L S. Recognizing actions by shape-motion prototype trees. In: 2009 IEEE 12th international conference on computer vision. IEEE; 2009, p. 444–51.
23. Blank M, Gorelick L, Shechtman E, Irani M, Basri R. Actions as space-time shapes. In: Null. IEEE; 2005, p. 1395–402.
24. Fe-Fei L, *etal*. A bayesian approach to unsupervised one-shot learning of object categories. In: Ninth IEEE proceedings of international conference on computer vision, 2003. IEEE; 2003, p. 1134–41.
25. Wolf L, Hassner T, Taigman Y. The one-shot similarity kernel. In: 2009 IEEE 12th international conference on computer vision. IEEE; 2009, p. 897–902.
26. Lake B, Salakhutdinov R, Gross J, Tenenbaum J. One shot learning of simple visual concepts. In: Proceedings of the annual meeting of the cognitive science society. 2011, p. 33.
27. Nair V, Hinton G E. Inferring motor programs from images of handwritten digits. In: Advances in neural information processing systems. 2006, p. 515–22.
28. Xu Z, Zhu L, Yang Y. Few-shot object recognition from machine-labeled web images. In: Computer vision and pattern recognition. 2017.