Learning Lexical Features of Programming Languages from Imagery Using Convolutional Neural Networks

Jordan Ott, Abigail Atchison, Paul Harnack, Natalie Best, Haley Anderson, Cristiano Firmani, Erik Linstead

Machine Learning and Assistive Technology Lab
Schmid College of Science and Technology
Chapman University
Orange, California

{ott109,atchi102,harna100,best120,ander427,firma103}@mail.chapman.edu,linstead@chapman.edu

ABSTRACT

We demonstrate the ability of deep architectures, specifically convolutional neural networks, to learn and differentiate the lexical features of different programming languages presented in coding video tutorials found on the Internet. We analyze over 17,000 video frames containing examples of Java, Python, and other textual and non-textual objects. Our results indicate that not only can computer vision models based on deep architectures be taught to differentiate among programming languages with over 98% accuracy, but can learn language-specific lexical features in the process. This provides a powerful mechanism for carrying out program comprehension research on repositories where source code is represented with imagery rather than text, while simultaneously avoiding the computational overhead of optical character recognition.

CCS CONCEPTS

- Computer systems organization → Neural networks; - Software and its engineering → Software libraries and repositories;

KEYWORDS

deep learning, convolutional neural networks, program syntax

ACM Reference Format:


1 INTRODUCTION

In recent years, the state-of-the-art in computer vision techniques have converged on deep learning solutions based on artificial neural networks. In particular, convolutional neural networks (CNNs) have demonstrated the ability to learn high-order features in imagery which can be leveraged for both supervised and unsupervised machine learning tasks. While this has led to a wide variety of applications in areas such as medical informatics and autonomy, the inherently textual nature of source code has left CNNs relatively unexplored in the software engineering community.

Despite the textual nature of source code, substantial volumes of software data remain embedded in images and video as part of technical tutorials available on the Internet. This data provides an opportunity for new directions in program comprehension research, particularly the ability of computer vision models to learn visual differences, such as syntax, among programming languages in the same way that human eyes perceive these features when looking at code. If computer vision models are capable of learning such features, these models can form the basis of new techniques for the automated understanding of code in images and videos that is not predicated on optical character recognition to first convert the data into text.

In this paper we apply, for the first time, CNNs to the task of learning lexical features from image-based representations of programs, focusing on Java and Python for a pilot study. Leveraging a training set of 17,500 hand-labeled images, we are able to achieve 98.73% cross-validated accuracy distinguishing Java and Python, and 92.50% accuracy distinguishing Java and Python from non-code data. Using class activation mapping (CAM) [16] as a visualization technique, we can see that our CNN models are learning low-level lexical features of programming languages as part of the training process, similar to cues a human uses to differentiate one language from another.

2 DATA

As a first step in exploring the suitability of CNNs for identifying lexical features of programming languages, we cultivated a corpus of 100 tutorials consisting of approximately 50 hours of video from YouTube. A diverse set of Integrated Development Environments (IDEs), text editors, font sizes, and text colors appear in the dataset. We focused on the Java and Python programming languages for the experiments described here. All 100 videos were downloaded to our server and segmented into a discrete image set by sampling at a rate of one frame per second. This resulted in 160,500 unlabeled images. These images were then filtered manually to exclude frames with code examples that were obstructed or obscured in any way, or contained handwritten code (such as on a whiteboard).
## 3 METHODS

Structured input data, such as images, lose their spatial relationships when passed through traditional fully-connected, feed-forward artificial neural networks (ANNs). This is problematic for applications such as computer vision since image features are comprised of groups of pixels. Convolutional Neural Networks represent an alternative ANN architecture that is able to maintain spatial relations between pixels by convolving the input space with a multidimensional weight matrix, commonly referred to as a filter. Training CNNs with backpropagation was first proposed by LeCun et al. [7]. 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 also builds translational invariance into the learned model, so that features can be recognized despite their specific location in the image.

In this paper, we leverage the VGG [11] network, a popular CNN architecture, to distinguish between Java and Python in video frames. The VGG network has a convenient architecture in which multiple convolutional operations occur in succession, followed by a max pooling layer, which has the effect of downsampling high-dimensional pixel spaces. After these layers have been repeated several times, a fully connected output layer, typically implementing a softmax function, is added. For our experiments, the VGG architecture was implemented in Python using the Keras API with a TensorFlow backend using two NVIDIA P100 GPUs with 16 GBs of memory and 3,584 CUDA cores each.

### 4 RESULTS

Convolutional neural networks were trained using 5-fold cross-validation for the two experiments detailed in Table 1. Each convolutional model took, on average, 2.5 hours per fold to train, for a total of 37.5 computing hours of training for all folds in all models. In practice, the overall time was decreased by training models in parallel by taking advantage of multiple GPUs on our deep learning server.

The mean accuracies of the 5-fold cross-validation experiments for each classifier are detailed below. A mean accuracy of 98.73% (median 98.75%) is achieved on the binary classification task of predicting Java versus Python typeset visible code. Including a third category of no code, yields an accuracy of 92.50% (median 92.60%). This accuracy was achieved when both Java and Python datasets contained only visible typeset code.

Figure 1 shows CAM results on correctly predicted Java code image frames. Additional CAM figures are available in the supplementary material. The heatmap produced by CAM 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. In Figure 1, the left column shows examples from the test set, while the right column shows the CAM overlaid on the corresponding test image. The first row shows an example of correctly predicted Java typeset code, while the second shows a Python image frame. Visual analysis of the CAM result images reveals the network’s preference towards Java and Python specific features such as method and class declarations, semicolons, and curly brackets. For example, the Java example in Figure 1 shows the network’s strong preference for curly brackets when predicting Java as the category. The Python image directly below it shows strong preference for the `def` keyword as well as Python’s indentation pattern. The results of Figure 1 show the network is capable of learning lexical and contextual features of image code frames.

To ensure our method can distinguish between Java and Python in homogeneous settings, we present the CAM results in Figure 2. Like Figure 1, the left column shows examples from the test set, while the right column shows the CAM overlaid on the corresponding test image. Both Java and Python code snippets are of the Quicksort algorithm. Each line is semantically the same, only differing in the languages appropriate syntax. The first row shows an example of correctly predicted Java typeset code, while the second shows a Python image frame. The Java CAM image shows the network identifies curly brackets, semicolons, backslashes for comments, and keywords such as public and private. The Python CAM image highlights then end of lines, indicating an absence of semicolons. These results show that convolutional networks, such

<table>
<thead>
<tr>
<th>Binary Classification</th>
<th>Training Set</th>
<th>Testing Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java vs Python</td>
<td>4,853</td>
<td>2,514</td>
</tr>
<tr>
<td></td>
<td>1,215</td>
<td>630</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Categorical Classification</th>
<th>Training Set</th>
<th>Testing Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java vs Python vs NC</td>
<td>4,853</td>
<td>2,514</td>
</tr>
<tr>
<td></td>
<td>5,495</td>
<td>5,495</td>
</tr>
<tr>
<td></td>
<td>1,215</td>
<td>630</td>
</tr>
<tr>
<td></td>
<td>1,370</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Average data set size of each cross validation fold.
Learning Lexical Features of Programming Languages from Imagery Using Convolutional Neural Networks ICPC ’18, May 27–28, 2018, Gothenburg, Sweden

Figure 1: CAM results on correctly predicted Java (top) and Python (bottom) code image frames. The left column shows the normal test image. The right column shows the CAM results overlaid on the test image.

as the ones used in this study, are capable of learning, visually, lexical differences between programming languages.

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 language’s syntax and not other circumstantial features contained in the images (IDE/text editor features). Additional CAM results, as well as the convolutional architecture used for the artificial neural networks in this study, are available in the supplementary material: https://github.com/mlat/icpc.

5 RELATED WORKS

The application of deep learning to the study of natural language represented as text was conducted in [2]. The authors leveraged a single CNN architecture to predict part-of-speech tags, chunks, named entity tags, as well as other semantic attributes given an input sentence. The authors in [6] use recurrent neural network grammars to identify and learn heads of phrases in order to determine the syntactic category of a natural language phrase. In [1], the authors propose a modular ANN architecture for lexical analysis of natural language in the form of a continuous input stream.

Digital image processing in the code recognition domain is outlined in [10]. In this study, the authors look to identify Java code in video frames through the application of OCR to candidate sub frames. In [14], the authors use language specific statistical modeling to identify code regions appearing across frames. In [9], deep-learning is applied to this domain through the application of convolutional neural networks to Java programming tutorials. This approach allows for a more scalable solution to video indexing while still maintaining accuracy. In this study, we extend this deep learning approach, focusing on building a single model that can differentiate between multiple languages while learning lexical features in the process.

The in-depth analysis of source code samples, although not in the form of images, has been explored in depth over the past decade. The study of vocabulary trends throughout Java software development is conducted in [8]. Others have sought to classify languages given source code samples. In [15], the authors present a maximum entropy classifier. The work in [12] employs a support vector machine based classifier while a statistical analysis of program features is conducted in [5]. In [4], a Multinomial Naive Bayes classifier is used and a modified Kneser-Ney discounting classifier is presented in [13]. The application of a CNN to language classification comes in [3]. This paper demonstrates the efficiency and accuracy of applying CNNs to the problem of classifying textual source code.

While previous work shares our goal of leveraging language features to classify programming languages, we believe our study
represents the first use of deep learning to do this using native images instead of text.

REFERENCES


