Artificial Intelligence Project
COSC 6368: Artificial Intelligence

Title of Project:
Data Augmentation in Program Domain

Group Members:

Md Rafiqul Islam Rabin
PeopleSoft ID: 1797648
Email: mrabin@central.uh.edu

Rubayat Jinnah
PeopleSoft ID: 1891217
Email:rjinnah@central.uh.edu

Date: July 20, 2020.

1 Introduction

Data is crucial in Artificial Intelligence (AI), to be specific in Deep Neural Networks. In practice, often one may have a limited sized or invariant dataset. However, complex models require complex datasets to generalize the model, avoid overfitting, and enhance performance. Nevertheless, a large number of parameters require a large number of examples. To create a complex large dataset, new data is augmented in the existing dataset. Researches have applied augmentation in the domain of image, audio, or text. However, applying augmentation in the program domain is very challenging for its complex structure. Therefore, we aim to study the program augmentation for generating new diverse augmented programs and retrain the model of programming task (i.e. code2vec [1]) to improve the generalizability for unseen data.

Considerations. This work studies the following two considerations.

- First, we study the task of code2vec from method-level to file-level to increase the area for augmentation. (Section 4.1).
- Second, we study the data augmentation in the program domain to improve the generalizability of the model. (Section 4.2).

2 Background and Survey

2.1 Definition

The process of generating new data to increase the diversity of the existing dataset is known as `Data Augmentation`. The new data is often called artificial data, augmented data, or synthetic data. Data augmentation has a number of domains and each domain has different techniques. Some popular domains of data augmentation are i) Image domain, ii) Audio domain, iii) Text domain, and iv) Program domain.

2.2 Survey of Research

In this section, we provide a comprehensive survey of research in AI (classical machine learning techniques, evolutionary techniques, and deep neural tech-

niques) that has been done in order to improve the generalizability of the model using data augmentation. Data augmentation domain, related field and techniques are shown below in the Table 1 at a glance.

Image Domain. In the field of computer vision, image processing, or medical data analysis, image data augmentation is very common. Since, data augmentation is relatively easier in the image domain, among all other domains, data augmentation is mostly implemented in the image domain. Initially, in 1997 data augmentation for image data was implemented by Ha & Bunke [6] to identify handwritten characters. Later in 2017, Antonio [4] demonstrated that robot vision can be improved up to 7% by augmenting data where a robot can recognize objects.

Data augmentation in image can be done in a variety of techniques [12]. A large dataset helps to train and classify images with higher accuracy. Usually, the image dataset is augmented by i) Position Augmentation and ii) Color Augmentation. Position Augmentation is achieved by the geometric transformation which includes Scaling, Cropping, Flipping, Padding, Rotation, Translation, Affine transformation. Color Augmentation is achieved by RGB transformation which includes changing Brightness, Contrast, Saturation, and Hue.

Audio Domain. Another popular domain of data augmentation is audio augmentation. In the field of speech recognition, data is augmented by signal manipulation. In 2015 Ko et al. [8] augmented audio data by accelerating or slowing the signal and showed an improvement of 4.3% across 4 different tasks. In 2013 Jaitly & Hinton [7] augmented audio data by noise injection and spectrogram modification and they observed 0.65% improvement on the test set.

Text Domain. Text data augmentation is a relatively complex process in comparison to image augmentation and audio augmentation. Although Natural Language Processing has some application of data augmentation, it is very limited. Because text data is very difficult to process and it is hard to generate realistic textual data. It is also hard to change a word or a phrase of a sentence maintaining the context. Also, not every word has a synonym. Until recently, a lexical substitution that consists to replace a word by its synonym using a thesaurus was the only widespread text augmentation technique where they did sentiment analysis [18].

Wang and Yang [14] used word embeddings for text augmentations, they proposed to use k-nearest-neighbor (KNN) and cosine similarity to find a similar word for replacement. Another approach for text augmentation is Back translation. Sugiyama and Yoshinaga [13] used English and Japanese language to translate and back translate which generates a new dataset for augmentation. They showed in their paper that using this augmentation process model learns to train based on context. Fadaee et al. [5] proposed TDA (Translation Data Augmentation) and showed that the machine translation model is improved by using text augmentation. They used contextualized word embeddings to replace target words rather than using static word embeddings. They used this text augmentation to test a machine translation model in data augmentation for low-resource neural machine translation. Random insertion, Random swap, and Random deletion are some of the other processes that are used to achieve text data augmentation [15].

Domain	**Field**	**Augmentation**	**Comment**
Image	Computer Vision, Image Processing, Image Analysis	Geometric Transformation, RGB Transformation	Widely applied augmentation
Audio	Speech Recognition	Accelerate or Slow speed, Noise injection, Spectogram modification	Popular augmentation processes
Text	Natural Language Processing	Thesaurus, Word Embedding, Back Translation	Complex and limited application
Program	Source Code, AST, Comment	Rename Field/Value, Add/Delete Code, Reorder/Swap Node,	Semantic-preserving transformation

Table 1: Summary of Domain-based Field and Augmentation Techniques.

Program Domain. The augmentation in source code has been occasionally studied by researchers because of its complexity in structure and scarcity in

strategy. Recently researches have shown their interest in testing and training robustness of model for programs under semantic-preserving program transformation [10]. They have generated adversarial programs by obfuscating variable [3], renaming identifiers [9,16,17] or fields [2,16], inserting dead code [2,9,11,16], changing constant [2] or boolean value [9, 11], reordering independent statements [2, 9, 11], and to name.

3 Model and Dataset

3.1 Code2Vec Model

The code2vec [1] is a TensorFlow 2 implemented deep neural network where the dataset is programming code and the target task is to predict the method's name given the method's body. For example, given the code: "`void f(int a, int b) {int t = a; a = b; b = t;}`", the model aims to predict the method's name as "`swap`". The code2vec uses a bag of abstract syntax tree (AST) paths where each path consists of a pair of terminal nodes and the path between those two nodes in the AST. Those path and terminal nodes are mapped into its vector embeddings which are learned jointly with other network parameters during training. The model encodes the AST path between leaf nodes and uses an attention mechanism to compute a learned weighted average of the path vectors in order to produce a single code vector. Finally, this code vector is used to predict the target method's name with a softmax-normalization between the code vector of method's body and the embedding of method's name.

3.2 SA Dataset

In this study, we need the raw `java` files amenable to data augmentation. We have collected the algorithm classification dataset, `SA-Dataset` [1]. This dataset contains a total of 855 raw Java files of 10 sorting problems from GitHub written in Java. A $55\% - 20\% - 25\%$ stratified partition of this dataset contains 475 training, 163 validation, and 216 test programs. The Appendix A shows the detail breakdown of the dataset over classes.

[1] `https://github.com/bdqnghi/ggnn.tensorflow#code-classification`

4 Methodology

4.1 Extending the Model to File Level

The code2vec model works in the method level where an input corresponds to a single method. However, when we write any program, we generally split it into multiple functions. For example, we write multiple functions such as main(), sort(), partition(), getPivot(), and swap() to implement the `quicksort` algorithm. Here, a specific function does not represent the `quicksort`, we need to consider the entire file of multiple functions for it [3]. To address this, we plan to extend the extractor part of the model to encode the entire file rather than a single method. This increases the scope of input program for augmentation.

The extractor of code2vec collects all path-context among leaf nodes in AST. A path-context consists of the source node, the destination node, and the intermediate path between source node and destination node. The extractor joins all path-context inside the method by anonymizing the method name that is used as a target. The path-context by code2vec [2] for the expression "`a = b;`" would be: [a, (NameExpr ↑ AssignExpr ↓ NameExpr), b]. The Appendix B shows an example of AST by code2vec. We modify the part of the extractor in the model to include all path-context of the entire file, rather than the path-context of a specific method. In other words, we join path-contexts of all methods with the actual method name to keep the dependency and connection among methods. Finally, we set the program category (i.e. quicksort) as the target, rather than any particular method name.

4.2 Data Augmentation in Program Domain

Data augmentation is very useful to increase the diversity of training data. In this section, we discuss several strategies for data augmentation in program domain. We have applied the JavaParser [3] tool for Java code analysis. We create a program augmentation tool, JavaAugmentation, to traverse the `CompilationUnit` of a program to access different parts and apply the following *five* program augmentation strategies to generate new diverse programs for

[2] `https://code2vec.org/`
[3] `https://github.com/javaparser/javaparser`

training model.

- `Function Augmentation` exchanges two independent functions inside a program. We find the `MethodCallExpr` nodes (M) in a program and swaps the position of two independent nodes.

- `Statement Augmentation` swaps the position of two independent statements inside a method. We find the `Statement` nodes (S) in a basic block and create pair of nodes (S_i, S_j) having no common identifiers among all statements $(S_i, ..., S_k, ..., S_j)$ where $i \leq k \leq j$, that ensures no dependency between statements.

- `Loop Augmentation` replaces a `ForStmt` node with a `WhileStmt`, and vice versa. We extract the condition and body from one node type, and initialize the other node type for replacement.

- `Switch Augmentation` replaces a `SwitchStmt` node with a sequence of `IfStmt` nodes, where each switch entry corresponds to each `IfStmt` node, and the default block statement corresponds to the `ElseStmt` node.

- `Binary Augmentation` exchanges two sides of the binary expression for following operators: `+, *, ||, &&, ==, !=, <, <=, >, >=`. We find the `BinaryExpr` nodes and extract the left expression, right expression, and operator. After that, we swap the left expression with the right expression. Note that we also replace the '`<`' operator with the '`>`' operator (and vice versa) to keep the semantic of condition.

Note that `Function Augmentation`, `Statement Augmentation`, and `Binary Augmentation` do not change any node in AST, but `Loop Augmentation` and `Switch Augmentation` replace few nodes in AST. However, this change does not alter the meaning of original programs, known as semantic-preserving. The Appendix C shows the example of program augmentation strategies.

4.3 Retraining Model with Augmented Programs

We train the code2vec model on both the original dataset (before data augmentation) and the augmented dataset (after data augmentation). The code2vec

Augmentation	Training	Validation	Test
Original	475	163	216
Function	1431	535	708
Statement	621	208	288
Loop	466	161	207
Switch	0	0	2
Binary	613	213	281
Total	3606	1280	1702

Table 2: Generated programs by different augmentation.

model uses F_1-`Score` as the evaluation metric to train and evaluate the model. We report the accuracy, score, learning curve for loss, and ROC curve on the unseen test set. [4]

$$Accuracy(A) = \frac{tp + tn}{tp + tn + fp + fn}$$
$$Precision(P) = \frac{tp}{tp + fp}, Recall(R) = \frac{tp}{tp + fn}$$
$$F_1 - Score = \frac{2}{P^{-1} + R^{-1}} = 2 * \frac{P * R}{P + R}$$

5 Result and Analysis

5.1 Generation of Augmented Programs

We have applied each program augmentation strategy on each data partition separately. Table 2 represents the number of generated new programs after program augmentation. The `Function Augmentation` generates the maximum number of new programs. Note that the `Switch Augmentation` does not generate new programs (shows 0 value in Table 2), because the training partition and validation partition do not have any sample program of `switch` case. We have merged all newly generated programs with the original dataset. This `augmented dataset` is key to increase the diversity of dataset for the training model.

[4] tp = True Positive, tn = True Negative, fp = False Positive, fn = False Negative.

5.2 Original Dataset vs. Augmented Dataset

We have trained the code2vec model on the original dataset and the augmented dataset separately up to 50 epochs and saved the model with results after each epoch. We have selected the best model based on the highest F_1`-Score` in the validation set, and loaded to predict the unseen test set.

5.2.1 Accuracy and F_1-Score

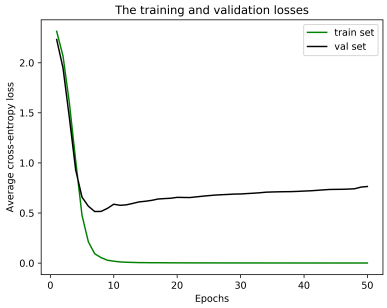
Table 3 shows that the model of the augmented dataset outperforms the model of the original dataset by 3.59% in the validation set, and 2.54% in the test set. We only report Accuracy (A) and F_1-Score as the metric values (A, P, R, F_1) are all same. This is because the data set contains `stratified` distribution of program class and the model calculates `weighted` metrics for each class by finding average weight by support.

Dataset	**Partition**	**Best Epoch**	**Accuracy (A)**	**Score (F_1)**
Original	Validation	7	87.11	87.11
Augmented	Validation	8	90.70	90.70
Original	Test	7	90.81	90.81
Augmented	Test	8	93.35	93.35

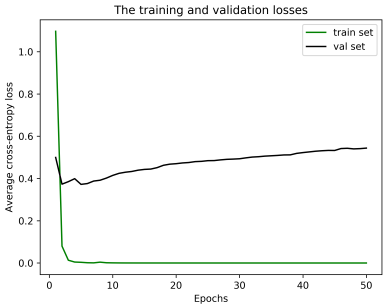
Table 3: Evaluation metrics on validation set and test set. Note that the target labels contain single token (algorithm name), and thus accuracy and score looks identical.

5.2.2 Learning Curve for Loss

Figure 1 shows the training and validation losses in a single plot (x-axis is the epochs, and the y-axis is the cross-entropy loss). The green line represents the training losses and the black line represents the validation losses. Initially, the training and validation losses are very high and then it decreases significantly over epochs. The plot shows that the model is overfitting around after 10 epochs as the validation losses increase gradually but training loss becomes stable.

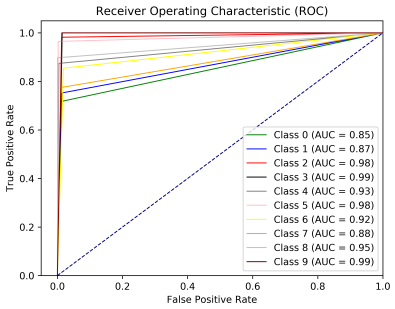


(a) Model of Original Dataset.

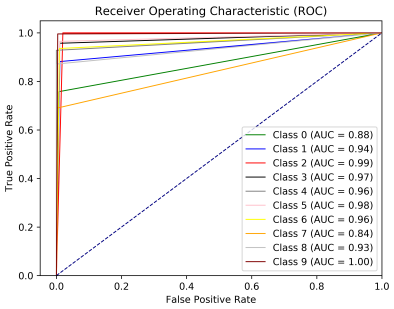


(b) Model of Augmented Dataset.

Figure 1: Learning curve for training and validation losses up to 50 epochs.



(a) Model of Original Dataset.



(b) Model of Augmented Dataset.

Figure 2: ROC curve for the unseen test dataset.

However, the best model we found in epoch 7 and 8, respectively. Therefore, the learning curve shows a good fit for the trained models.

5.2.3 ROC Curve for Separation Capability

Figure 2 shows the ROC (Receiver Operating Characteristic) curve that is another performance measurement for classification which indicates the capability of separation between classes. The AUC (Area Under Curve) value near 1 means the model has good measure of separability, where 0 represents the worst, and when AUC is 0.5, it means model has no class separation capacity. The plot shows that the average AUC (Area Under Curve) value for the model of the augmented dataset is comparatively higher than the model of the original dataset.

Also, most of the cases, the model of the augmented dataset dominates over the model of the original dataset for each class.

6 Conclusion

In this study, we have investigated the data augmentation strategy in the program domain with an extended deep neural network, code2vec for file level. In the beginning, we survey state-of-the-art data augmentation techniques that aim to increase the diversity of the existing dataset by adding newly augmented data without actually collecting data. However, the augmentation approach is very challenging and rarely explored in the program domain. Therefore, we have studied several program augmentation strategies and conducted a pilot study with a popular neural model (code2vec) and dataset (SA) of programs. To increase the area for augmentation, we have also extended the focus of the model into file-level from method-level by modifying the extractor part. Finally, we have trained the model on the original dataset and the augmented dataset, separately. The results from accuracy, learning curve of loss, and ROC curve suggest that the model can benefit from the augmented dataset.

Our implementation can be found at https://github.com/mdrafiqulrabin/ai-program-augmentation/. In future work, we plan to apply a large scale analysis on the various classification models, tasks and datasets.

References

[1] Alon, U., Zilberstein, M., Levy, O., and Yahav, E. code2vec: Learning distributed representations of code. *arXiv preprint arXiv:1803.09473* (2018).

[2] Bielik, P., Raychev, V., and Vechev, M. Phog: probabilistic model for code. In *International Conference on Machine Learning* (2016), pp. 2933–2942.

[3] Compton, R., Frank, E., Patros, P., and Koay, A. Embedding java classes with code2vec: Improvements from variable obfuscation. *arXiv preprint arXiv:2004.02942* (2020).

[4] D'Innocente, A., Carlucci, F. M., Colosi, M., and Caputo, B. Bridging between computer and robot vision through data augmentation: a case study on object recognition. In *International Conference on Computer Vision Systems* (2017), Springer, pp. 384–393.

[5] Fadaee, M., Bisazza, A., and Monz, C. Data augmentation for low-resource neural machine translation. *arXiv preprint arXiv:1705.00440* (2017).

[6] Ha, T. M., and Bunke, H. Off-line, handwritten numeral recognition by perturbation method. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 5 (1997), 535–539.

[7] Jaitly, N., and Hinton, G. E. Vocal tract length perturbation (vtlp) improves speech recognition. In *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language* (2013), vol. 117.

[8] Ko, T., Peddinti, V., Povey, D., and Khudanpur, S. Audio augmentation for speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association* (2015).

[9] Rabin, M. R. I., and Alipour, M. A. Evaluation of generalizability of neural program analyzers under semantic-preserving transformations. *arXiv preprint arXiv:2004.07313* (2020).

[10] Rabin, M. R. I., Wang, K., and Alipour, M. A. Testing neural program analyzers. In *34th IEEE/ACM International Conference on Automated Software Engineering (Late Breaking Research-Track)* (2019).

[11] Ramakrishnan, G., Henkel, J., Wang, Z., Albarghouthi, A., Jha, S., and Reps, T. Semantic robustness of models of source code. *arXiv preprint arXiv:2002.03043* (2020).

[12] Shorten, C., and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data 6*, 1 (2019), 60.

[13] Sugiyama, A., and Yoshinaga, N. Data augmentation using back-translation for context-aware neural machine translation. In *Proceedings of the Fourth Workshop on Discourse in Machine Translation (DiscoMT 2019)* (2019), pp. 35–44.

[14] Wang, W. Y., and Yang, D. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using# petpeeve tweets. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (2015), pp. 2557–2563.

[15] Wei, J., and Zou, K. Eda: Easy data augmentation techniques for boost-

ing performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).

[16] Yefet, N., Alon, U., and Yahav, E. Adversarial examples for models of code. *arXiv preprint arXiv:1910.07517* (2019).

[17] Zhang, H., Li, Z., Li, G., Ma, L., Liu, Y., and Jin, Z. Generating adversarial examples for holding robustness of source code processing models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 1169–1176.

[18] Zhang, X., and LeCun, Y. Text understanding from scratch. *arXiv preprint arXiv:1502.01710* (2015).

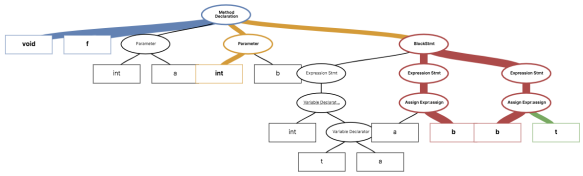
Appendix A Breakdown of SA-Dataset

Class Label	Sorting Type	Number of Program
0	Bubble-Sort	98
1	Heap-Sort	80
2	Merge-Sort	86
3	Radix-Sort	79
4	Shell-Sort	83
5	Bucket-Sort	84
6	Insertion-Sort	104
7	Quick-Sort	72
8	Selection-Sort	96
9	Topological-Sort	73

Table 4: Number of instance over class in SA-dataset.

Appendix B Path-Context of Code2Vec

```
void f (int a, int b) {
    int t = a;
    a = b;
    b = t;
}
```



(a) Program. (b) AST.

Figure 3: Example of a sample program and AST for '`swap`' method.

Appendix C Type of Program Augmentation

```
public class Func {
    public int get_a () {
        int a = 2;
        return a;
    }
    public int get_b () {
        int b = 3;
        return b;
    }
    public void main() {
        a = get_a();
        b = get_b();
        int c = a + b;
    }
}
```

```
public class Func {
    public int get_b () {
        int b = 3;
        return b;
    }
    public int get_a () {
        int a = 2;
        return a;
    }
    public void main() {
        a = get_a();
        b = get_b();
        int c = a + b;
    }
}
```

Figure 4: Example of `Function Augmentation`.

```
public class Stmt {
    public void main() {
        a = get_a();
        b = get_b();
        int c = a + b;
    }
}
```

```
public class Stmt {
    public void main() {
        b = get_b();
        a = get_a();
        int c = a + b;
    }
}
```

Figure 5: Example of `Statement Augmentation`.

```
public class Loop {
    public void main() {
        int s = 0;


        for (int i=0; i<10; i++) {
            s = s + i * s;

        }


    }
}
```

```
public class Loop {
    public void main() {
        int s = 0;
        {
            int i=0;
            while (i<10) {
                s = s + i * s;
                i++
            }
        }
    }
}
```

Figure 6: Example of `Loop Augmentation`.

```
public class Switch {
    public int main() {
        a = get_a();
        b = get_b();
        char op = get_op();
        switch(op) {
            case '+':
                return a + b;
                break;
            case '-':
                return a - b;
                break;
        }
    }
}
```

```
public class Switch {
    public int main() {
        a = get_a();
        b = get_b();
        char op = get_op();

        if (op == '+') {
            return a + b;

        } else if (op == '-') {
            return a - b;

        }
    }
}
```

Figure 7: Example of `Switch Augmentation`.

```
public class Binary {
    public void main() {
        a = get_a();
        b = get_b();
        int c = a + b;
    }
}
```

```
public class Binary {
    public void main() {
        a = get_a();
        b = get_b();
        int c = b + a;
    }
}
```

Figure 8: Example of `Binary Augmentation`.