SAND2020-10141R

Using Neural Architecture Search for Improving Software Flaw Detection in Multimodal Deep Learning Models

Alexis Cooper acoope@sandia.gov
Xin Zhou xzhou1@sandia.gov
Scott Heidbrink sheidbr@sandia.gov
Daniel M. Dunlavy[†] dmdunla@sandia.gov
Sandia National Laboratories
Albuquerque, NM 87123, USA

Abstract

Software flaw detection using multimodal deep learning models has been demonstrated as a very competitive approach on benchmark problems. In this work, we demonstrate that even better performance can be achieved using neural architecture search (NAS) combined with multimodal learning models. We adapt a NAS framework aimed at investigating image classification to the problem of software flaw detection and demonstrate improved results on the Juliet Test Suite, a popular benchmarking data set for measuring performance of machine learning models in this problem domain.

Keywords: multimodal deep learning, neural architecture search, software flaw detection

1. Introduction

Most current approaches for software flaw detection rely on analysis of a single representation of a software program (e.g., source code or program binary compiled in a specific way for a specific hardware architecture). Recent work using multiple software representations and multimodal deep learning illustrates the benefits of leveraging both source and binary information in detecting flaws [5]. However, when using deep learning models, determining the most effective neural network architecture can be a challenge. Neural architecture search (NAS) is one way to perform an automated search across many different neural network architectures to find improved model architectures over manually-designed ones. In this work, we use a gradient-based NAS method that leverages a differentiable architecture sampler (GDAS) [2], which was identified as the best NAS method across 10 popular approaches when applied to image classification problems [3].

The remainder of this report is organized as follows. In Section 2, we provide an overview of the multimodal deep learning and NAS methods used to create flaw detection models.

(†) Corresponding author.

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

In Section 3, we define the set of experiments conducted to assess performance of these models over the baseline of not using NAS. In Section 4, we present the results of these experiments on a standard benchmark data set used in flaw detection research. And, finally, in Section 5, we summarize our conclusions and provide suggestions for future work in this area.

2. Methods

In this section, we describe the Joint Autoencoder (JAE) multimodal deep learning model for software flaw detection [5] and the cell-based neural architecture search (NAS) approach used to determine an optimal architecture for that model.

2.1 Multimodal Deep Learning for Software Flaw Detection

The neural network architecture selected for these experiments is an early fusion multimodal learning model called Joint Autoencoder (JAE) [4]. JAE was originally developed for learning multiple tasks simultaneously based on sharing features that are common to all tasks. Figure 1(a) illustrates the architecture of the original JAE model, which contains 2 encoder/decoder components per modality and a single mixing component that combines the output from one of the encoders associated with each modality. The components that do not interact with the mixing component are referred to as *private branches* [4]. Note that each of the components depicted in the image (i.e., each box in the image) can contain one or more traditional neural network layers. Recently, an adaptation of the JAE model, referred to here as the JAE Classifier Model, was developed for classifying software functions as to whether or not they contain flaws/bugs [5]. Figure 1(b) illustrates the architecture of the JAE Classifier Model, where we remove the decoders and use a linear layer to concatenate the outputs from previous layers. In the JAE Classifier Model, we use one or more linear layers with LeakyReLU activation for encoders and the mixing components. In the first linear layer, the number of input features will be the total length of two private branch encoders plus the number of output features from mixing component, and the number of output features is fixed as 50. In the final linear layer, a classifier layer is used, mapping 50 input features to the number of classes. In the flaw detection models used here, we use two classes, *flawed* and *not flawed*.

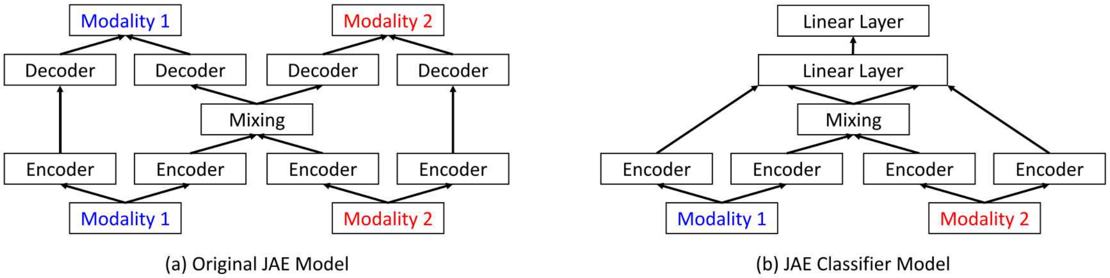


Figure 1: JAE Structure

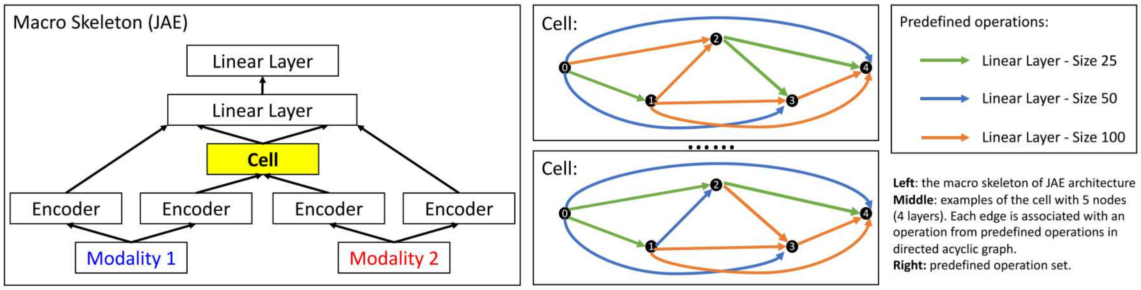
2.2 Neural Architecture Search

The JAE architectures described in the previous section were designed manually and thus may not be optimal for the learning tasks to which they are applied. To address this potential issue, we leverage a Neural Architecture Search (NAS) strategy to determine an optimal architecture for the flaw detection task. The specific form of NAS we employ here is based on cell-based search, in which a cell represents a portion of the architecture and is defined using a densely-connected directed acyclic graph (DAG) [3]. The edges of the DAG represent architecture layers and the nodes represent sums of the feature maps output from each of those layers. The search is performed over a set of operations (i.e., network layers) and the weights associated with those operations. Optimization of the cell structure and weights is performed within each iteration of the overall model training.

In this work, we define the macro skeleton, i.e., the full NAS architecture, as the JAE Classifier Model and the cell as the mixing layer with that model. Figure 2 illustrates the macro skeleton architecture (left), example DAG instances of the cell (center), and the cell operations used in our work (right). As noted in the image, the cell operations consist of single linear layers of sizes 25, 50, and 100 (i.e., the number of nodes in the layer). Details of the interpretation of the cell examples as sums of the feature maps of the operations can be found in [2].

We adapt the Automated Deep Learning (AutoDL) NAS comparison framework[1]., which implements the NAS-BENCH-201 [3] image classification benchmark, for use with our flaw detection classification problem. As recommended in the NAS-BENCH-201 experiments on images and confirmed in preliminary experiments with the JAE Classifier Model, we use the GDAS search strategy [2] in the work presented here. GDAS is a gradient-based search method using differentiable architecture sampler to optimize the cell search, and it has been demonstrated to be one of the more efficient NAS techniques that relies on more than simple random sampling for the cell search.

Optimization of the weights in the cell layers is performed using stochastic gradient descent (SGD) [8] and the overall macro skeleton architecture model fitting is performed using the ADAM optimizer [6], both as implemented in the AutoDL framework.



Left: the macro skeleton of JAE architecture
Middle: examples of the cell with 5 nodes (4 layers). Each edge is associated with an operation from predefined operations in directed acyclic graph.
Right: predefined operation set.

Figure 2: JAE Structure

[1] https://github.com/D-X-Y/AutoDL-Projects

3. Experiments

In this section, we describe the experiments we performed to answer the following questions:

- Are there differences between handcrafted JAE structure and selected structure from NAS?
- Are there improvements on flaw detection performance after implementing NAS?

3.1 Data

As we are measuring potential improvements when using NAS on the JAE Classifier Model, we use the same subset of the Juliet Test Suite data [7] from the software flaw detection experiments performed in [5]. The Juliet Test Suite [7] is a collection of test cases in the C/C++ language, providing pairs of functions with and without software flaws. The test cases laws are organized into collections based on the Common Weakness Enumeration (CWEs) of the specific flaws exhibited in each function. Table 1 lists the test case CWE collections used in this work. This set of test cases represents a wide range of the types of flaws found in real-world software systems. We use the features extracted from this data as defined in [5].

In our experiments, we split each CWE collection into three data sets: 80% train, 10% validation, and 10% test. For cell search, we use the train and validation data sets to search for the best cell.

CWE	Flaw Description	# Flawed	# Not Flawed
121	Stack Based Buffer Overflow	6346	16868
190	Integer Overflow	3296	12422
369	Divide by Zero	1100	4142
377	Insecure Temporary File	146	554
416	Use After Free	152	779
476	NULL Pointer Dereference	398	1517
590	Free Memory Not on Heap	956	2450
680	Integer to Buffer Overflow	368	938
789	Uncontrolled Mem Alloc	612	2302
78	OS Command Injection	6102	15602

Table 1: Juliet Test Suite Data Summary

3.2 Methods used in Experiments

We compare flaw detection results using the JAE Classifier Model and application of the GDAS to the cell-based macro skeleton described in the previous section. The manually-designed JAE Classifier Model used a mixing component with a single linear layer consisting of 50 nodes, and we refer to this model as the *JAE-Mixing-50* model. In our experiments, we also investigated the use of a larger layer of size 100, and we refer to that model here as the *JAE-Mixing-100* model. The GDAS-based model is referred to here as the NAS-GDAS-JAE model.

3.3 Measurements used in Comparing Methods

For each of the Juliet Test Suite CWE collections, we performed $N \times 2$ cross validation [1] with $N = 5$. We use this form of cross validation as it provide a pessimistic estimate of the generalization error; when training models for operational use, we often use more than 50% of our training data to fit the final model. We use class-averaged accuracy—the average of the accuracies of instances from each class, normalized by the size of each class—to adjust for the skew in the sizes of the *flawed* and *not flawed* instance (see Table 1 for details). This approach addresses skew by not favoring classification results from either of the classes when they are not equal in size. For each method, we compute and report the sample mean and sample standard deviation of the class-averaged accuracy results for each method on each CWE collection.

3.4 Cell Structure Optimization

As mentioned earlier, in the NAS-GDAS-JAE model, cell search is performed using SGD optimization. The specific parameters used in the AutoDL implementation of SGD are provided in Table 2.

Parameter	**Value**
scheduler	cos
LR	0.0005
eta_min	0.001
epochs	100
optim	SGD
decay	0.000001
momentum	0.9
nesterov	1
criterion	Softmax
batch_size	32

Table 2: NAS-GDAS-JAE Parameters for SGD Cell Search

3.5 Cell Structure Representation

The result of the cell search in the NAS-GDAS-JAE model is a DAG representing several linear layers of different sizes (based on our defined cell operations). The AutoDL framework in which we implemented NAS-GDAS-JAE represents a DAG instance using a string to define the specific cell operations and sums of feature maps. Figure 3 illustrates the string output of an example DAG, which is

```
|100~0| + |50~0|100~1| + |25~0|50~1|50~2| + |25~0|100~1|25~2|50~3|  .
```

This summands in the string represent the sums of the feature maps associated with different cell operations. Each sum is defined inside the “| |” delimiters, where each cell operation and the edge source node is listed. For example, the summand in the example above of "|25˜0|50˜1|50˜2|" represents the sum of the feature maps of three cell operations (i.e., linear layers) at node 3 as depicted in the image—the green edge (size 25) from node 0, the blue edge (size 50) from node 1, and the blue edge (size 50) from node 2.

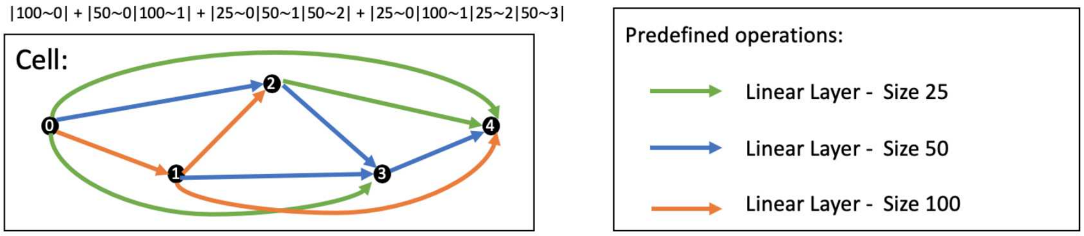


Figure 3: Example Cell Structure in the NAS-GDAS-JAE Model

4. Results

In this section, we present the results of our experiments leveraging multimodal learning models and neural architecture search to address the question of software flaw detection.

4.1 Optimized Cell Structure of NAS-GDAS-JAE Models

The optimized cell structure of the NAS-GDAS-JAE models for each of the Juliet Test Suite data sets can be found in Table 3. Note that none of the final cell structures across the difference data sets are the same.

Table 3: GDAS-JAE Search Results

<table>
<tr><th>CWE</th><th>Cell Structure (string representation of DAG)</th></tr>
<tr><td>121</td><td>|100~0| + | 50~0|100~1| + | 25~0| 50~1| 50~2| + | 25~0|100~1| 25~2| 50~3|</td></tr>
<tr><td>190</td><td>| 50~0| + |100~0| 25~1| + | 25~0| 25~1| 50~2| + |100~0| 50~1|100~2| 25~3|</td></tr>
<tr><td>369</td><td>| 25~0| + | 25~0|100~1| + | 25~0|100~1| 25~2| + | 50~0| 25~1| 25~2|100~3|</td></tr>
<tr><td>377</td><td>| 50~0| + | 25~0| 25~1| + | 50~0| 25~1|100~2| + |100~0|100~1| 50~2| 25~3|</td></tr>
<tr><td>416</td><td>| 25~0| + | 50~0|100~1| + | 50~0|100~1|100~2| + | 25~0|100~1|100~2| 50~3|</td></tr>
<tr><td>476</td><td>|100~0| + |100~0| 50~1| + | 25~0| 50~1| 25~2| + | 50~0| 25~1| 50~2| 50~3|</td></tr>
<tr><td>590</td><td>|100~0| + | 50~0| 25~1| + | 50~0|100~1|100~2| + |100~0| 25~1| 50~2|100~3|</td></tr>
<tr><td>680</td><td>|100~0| + |100~0| 50~1| + | 50~0|100~1|100~2| + | 50~0| 50~1| 50~2| 25~3|</td></tr>
<tr><td>78</td><td>|100~0| + | 50~0|100~1| + |100~0|100~1| 50~2| + |100~0| 50~1| 25~2| 50~3|</td></tr>
<tr><td>789</td><td>| 25~0| + | 25~0| 25~1| + | 25~0| 50~1|100~2| + | 50~0| 50~1|100~2|100~3|</td></tr>
</table>

The differences in cell structures may be due to the fact that the cell search is a global optimization problem, but the SGD method is only guaranteed to find a local optimizer. Or this may be due to the differences between the data associated with the different flaw types. More work is needed to better understand the source for these differences. To illustrate some of the differences, we present plots of the convergence behaviors of the cell search (search) and macro skeleton architecture (eval) optimizations in Appendix A. Over 100 epochs, we see a wide range of behaviors, maximum accuracy values achieved, and search/eval differences across the various data sets. More work is needed to better understand how these convergence behaviors impact the flaw detection results in general.

4.2 Flaw Detection Results

Table 4 shows the flaw detections results using the three models descried above. The two *JAE-Mixing-N* models (with $N = 50$ and $N = 100$) are considered baselines for the NAS-GDAS-JAE model, as they use the manually-designed architecture described in previous results [5]. The results listed in the table are the sample means and sample standard standard deviations of the class averaged accuracy per Juliet Test Suite data set. The boldfaced results indicate the best mean class-averaged accuracy for each data set (i.e., per row). Note that many of the differences between the means are not separated by more than a single sample standard deviation (across methods/columns), and thus the improvements using NAS may not be statistically significant. More work is need to determine if these improvements generalize and are statistically significant.

Table 4: Sample means and standard deviations of class averaged accuracy using 5×2 cross validation (boldfaced results are best across methods for each data set)

CWE	**JAE-Mixing-50**	**JAE-Mixing-100**	**NAS-GDAS-JAE**
121	0.9972±0.0009	**0.9975**±0.0008	0.9970±0.0012
190	0.9867±0.0068	**0.9907**±0.0059	0.9884±0.0067
369	0.9485±0.0206	0.9500±0.0203	**0.9703**±0.0220
377	0.9309±0.0614	0.9285±0.0420	**0.9514**±0.0410
416	0.9074±0.0620	0.9359±0.0471	**0.9468**±0.0400
476	0.9991±0.0019	**1.0000**±0.0000	**1.0000**±0.0000
590	**1.0000**±0.0000	**1.0000**±0.0000	**1.0000**±0.0000
680	0.9344±0.0139	0.9356±0.0115	**0.9417**±0.0197
78	0.9398±0.0110	0.9360±0.0143	**0.9427**±0.0155
789	0.9672±0.0201	0.9630±0.0183	**0.9683**±0.0215

5. Conclusions

In this work, we implemented a cell-based neural architecture search strategy to improve upon a manually-designed multimodal learning model for software flaw detection. Our results indicate that NAS leads to improved multimodal models that are specific to the software data being analyzed. These preliminary results provide a starting point for leveraging NAS for such a problem, as there are many open questions that still need to be addressed. In the work presented here, we used a cell that replaces only a small part of the JAE Classifier Model from [5]. However, larger, more complicated cells could lead to more pronounced improvements, but this would come at increased optimization and training cost as well. Determining the trade-offs between cell complexity and computational cost could be a useful research activity.

References

[1] Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[2] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. arxiv:1910.04465, 2019.

[3] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[4] Baruch Epstein, Ron Meir, and Tomer Michaeli. Joint autoencoders: a flexible meta-learning framework. https://openreview.net/forum?id=S1tWRJ-R-, 2018.

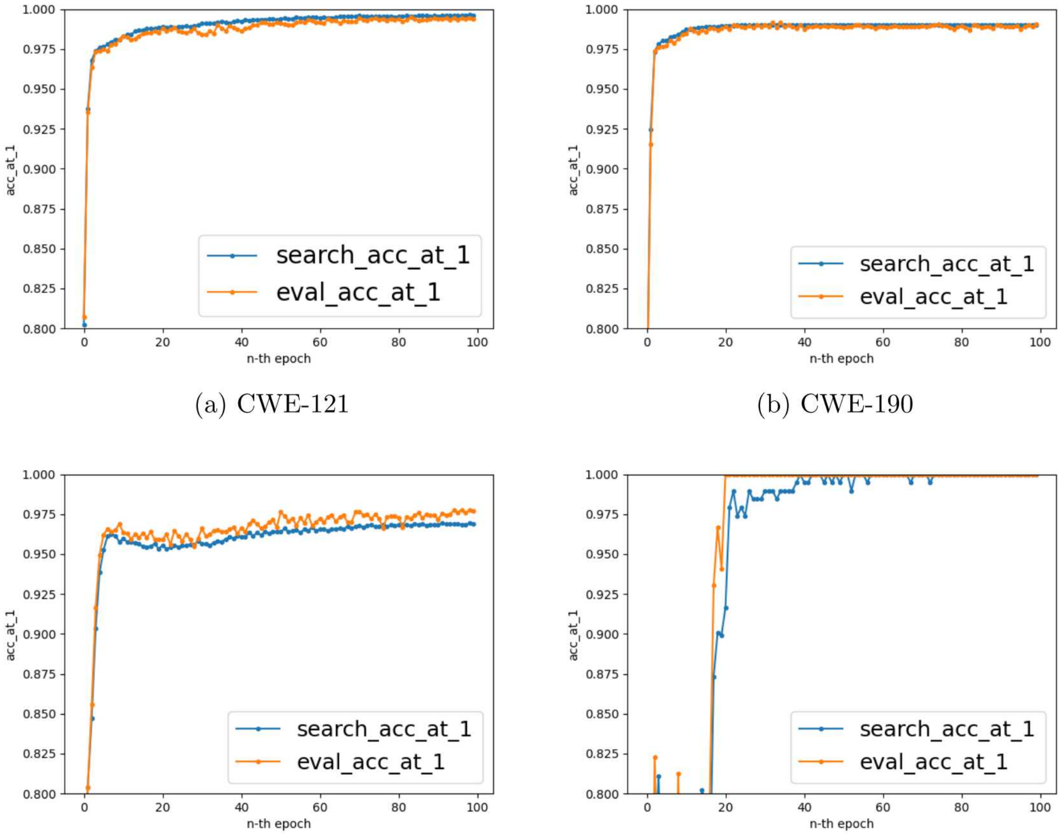
[5] Scott Heidbrink, Kathryn N. Rodhouse, and Daniel M. Dunlavy. Multimodal deep learning for flaw detection in software programs. arXiv:2009.04549, 2020.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

[7] NIST. Juliet test suite for C/C++ v1.3. https://samate.nist.gov/SRD/testsuite.php, 2017.

[8] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv:1609.04747, 2016.

Appendix A. NAS-GDAS-JAE Cell Search Results



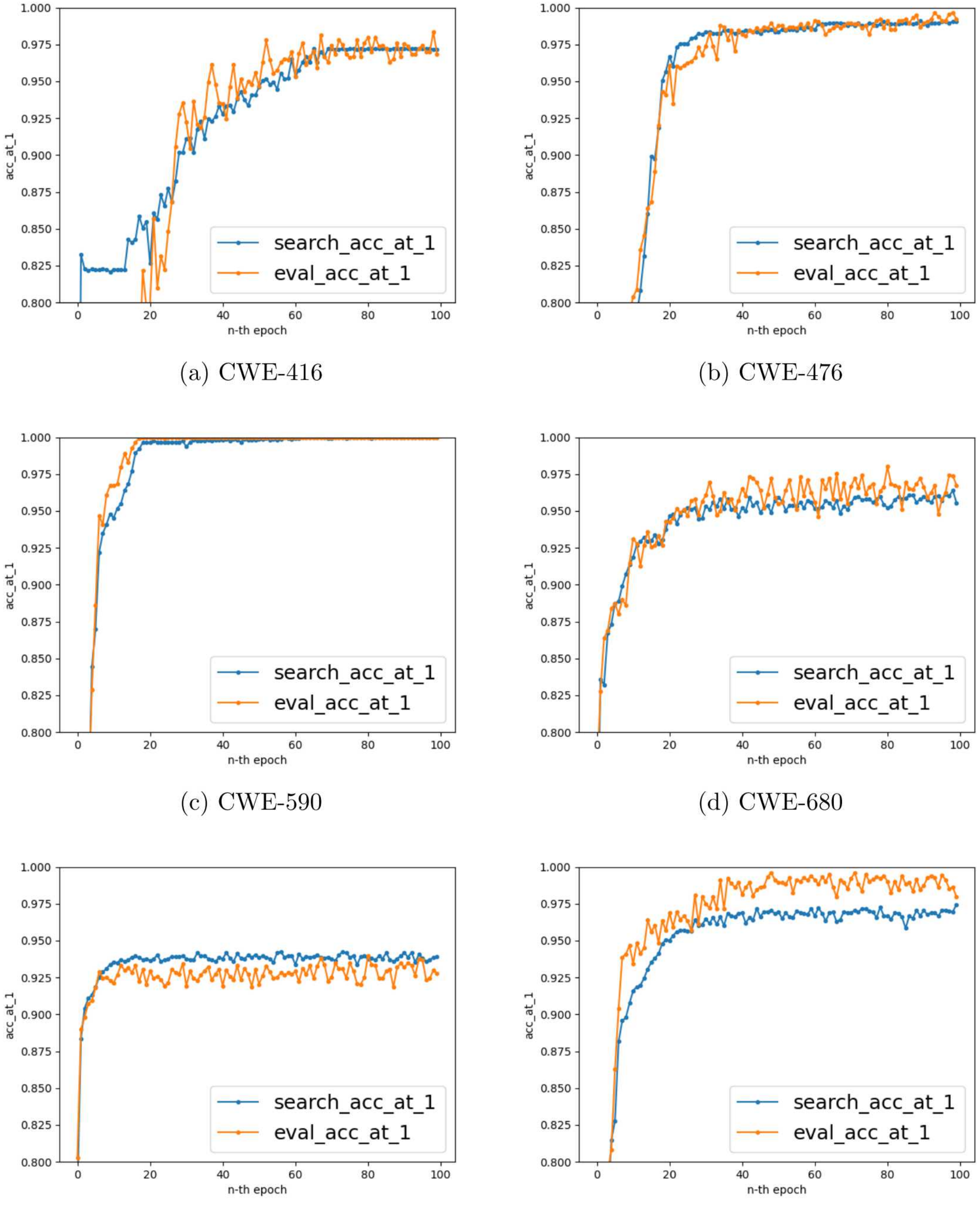
(a) CWE-121

(b) CWE-190

(c) CWE-369

(d) CWE-377

Figure 4: NAS-GDAS-JAE Cell Search Results - Part 1



(a) CWE-416

(b) CWE-476

(c) CWE-590

(d) CWE-680

(e) CWE-78

(f) CWE-789

Figure 5: NAS-GDAS-JAE Cell Search Results - Part 2