Predicting Unit-Test Scores for Graphic Representations of Simple Computer Programs

Richard Davis
Spring 2017, CS231n

Introduction

- As enrollment in computer science courses grows, automated methods of grading student code submissions and giving feedback are necessary. However, it has proven difficult to represent code in a way that can be used in machine learning.
- Prior methods represent code using abstract-syntax trees (Huang et al., 2013), as collections of modifications that do not change functionality (Nguyen et al., 2014), and recursive neural network embeddings of Hoare Triples (Piech et al., 2015).
- This project introduces a new program embedding: images created from parsing the abstract-syntax tree. These images resemble programs written in Scratch (Resnick et al., 2009) or Snap (Harvey et al., 2013).
- To test the efficacy of this embedding method, we use a variety of convolutional neural network (CNN) architectures to predict the unit-test scores.
- Some CNN architectures significantly outperform a simple baseline, showing that this embedding method captures information about the program that can be learned by a CNN.

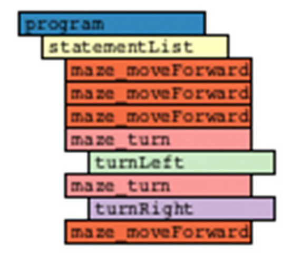
Research Questions

- Can we transform simple programs into images in a way that captures information about the program functionality? Can this information be learned by a CNN?

Data

- The raw data consist of student code submitted to the code.org website.
- The raw data contain over 100,000 unique code submissions from 762,974 unique users.
- I transformed the ASTs from these code submissions into images.
- The ASTs were categorized by their unit-test scores. These scores were converted into 6 categories.
- Code submissions were for two problems: HOC4 and HOC18.
- HOC18 was significantly more difficult than HOC4.
- HOC18 required the use of conditionals and loops, unlike HOC4.
- The images were RGB 250x250. The colors for each block were chosen using ColorBrewer2 for maximum contrast.

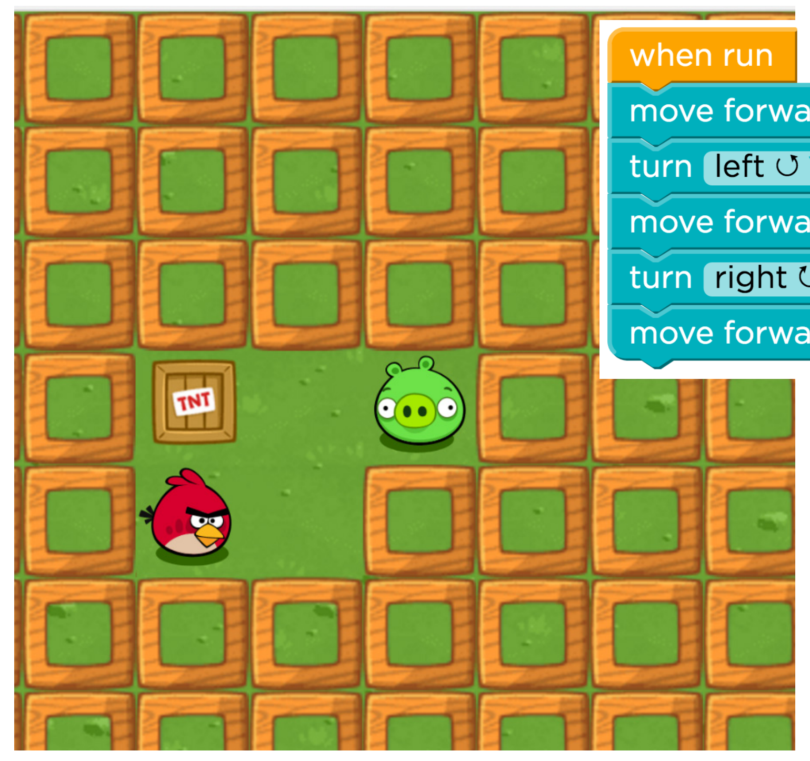
```
"id": "8",
"children": [{
    "id": "7",
    "children": [
        {
            "id": "0",
            "type": "maze_moveForward"
        },
        {
            "id": "2",
            "children": [{
                "id": "1",
                "type": "turnLeft"
            }],
            "type": "maze_turn"
        },
        {
            "id": "3",
            "type": "maze_moveForward"
        },
        {
            "id": "5",
            "children": [{
                "id": "4",
                "type": "turnRight"
            }],
            "type": "maze_turn"
        },
        {
            "id": "6",
            "type": "maze_moveForward"
        }
    ],
    "type": "statementList"
}],
"type": "program"
```



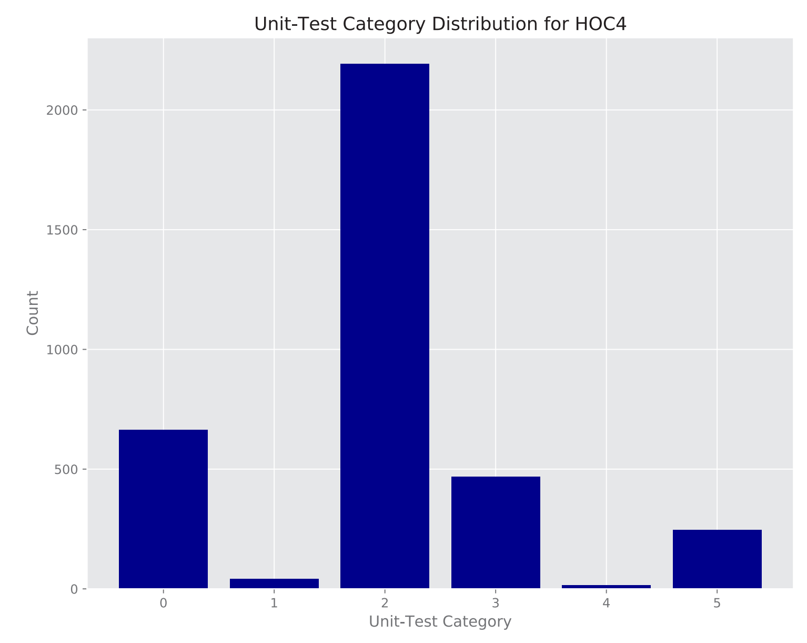
Architectures

- Simple baseline (predict most common class)
- Vanilla CNN
- ResNet-18, ResNet-34, ResNet-50, ResNet-101
- VGG-11, VGG-13, VGG-16
- SqueezeNet 1.1

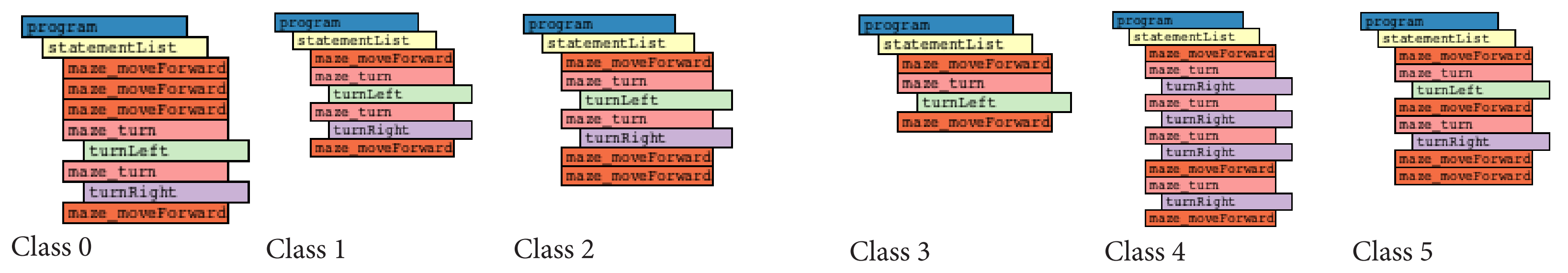
The HOC4 Problem



- The HOC4 maze and solution

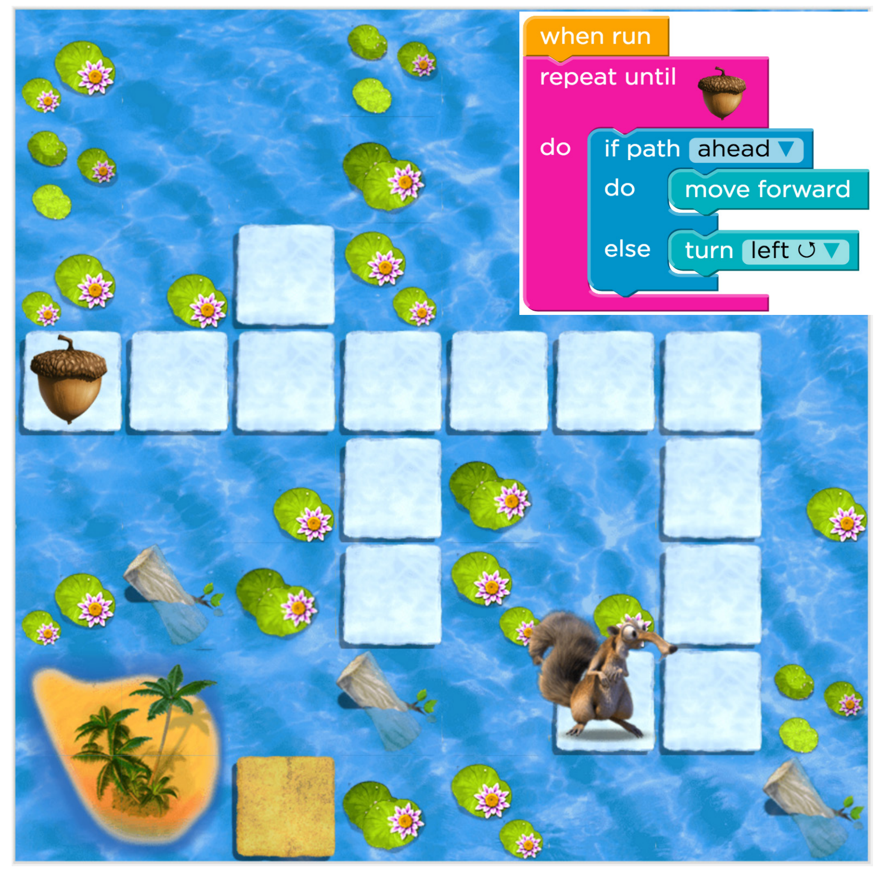


- The distribution of categories in HOC4

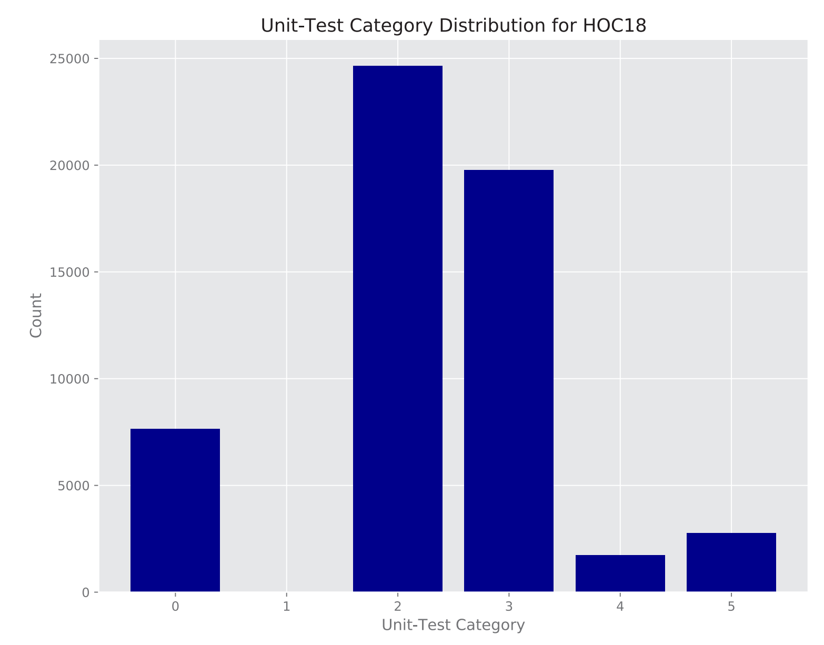


- Examples of each category in the HOC4 data

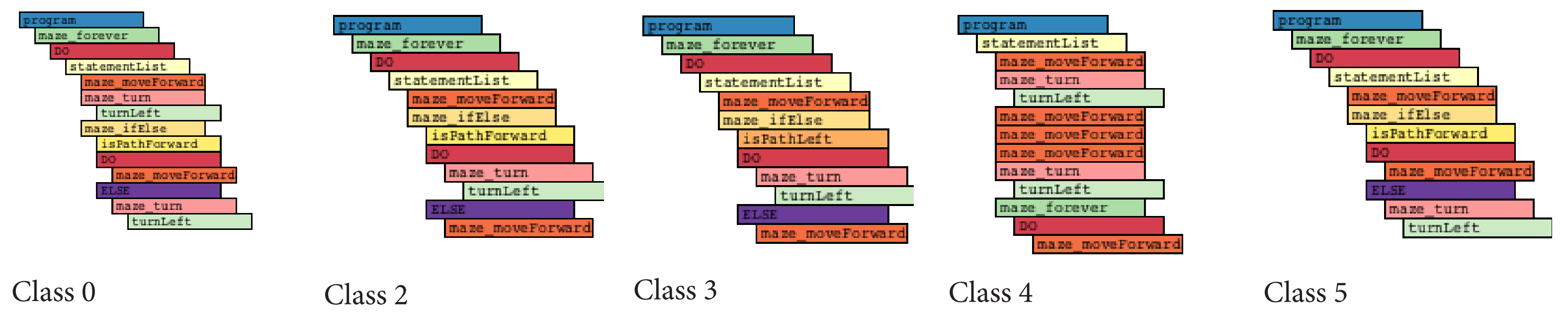
The HOC18 Problem



- The HOC18 maze and solution



- The distribution of categories in HOC18

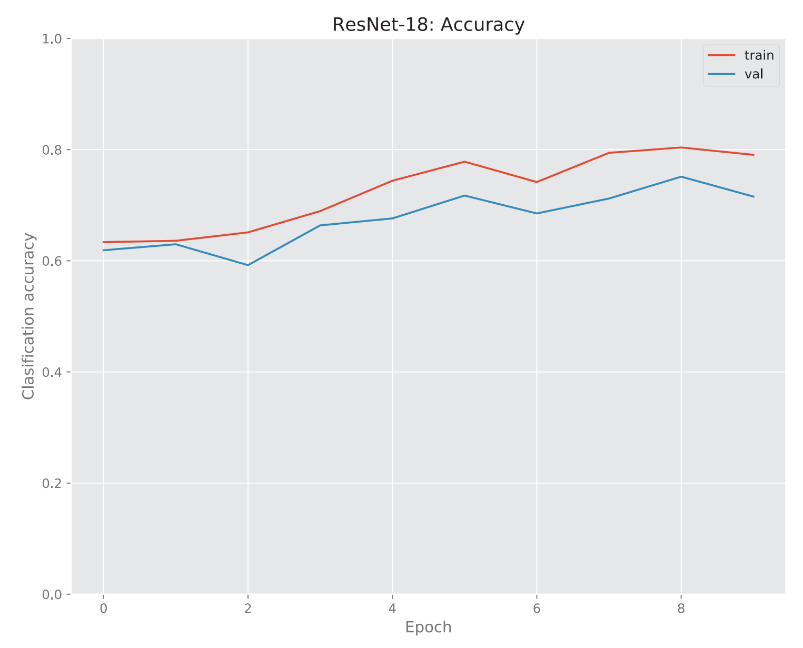


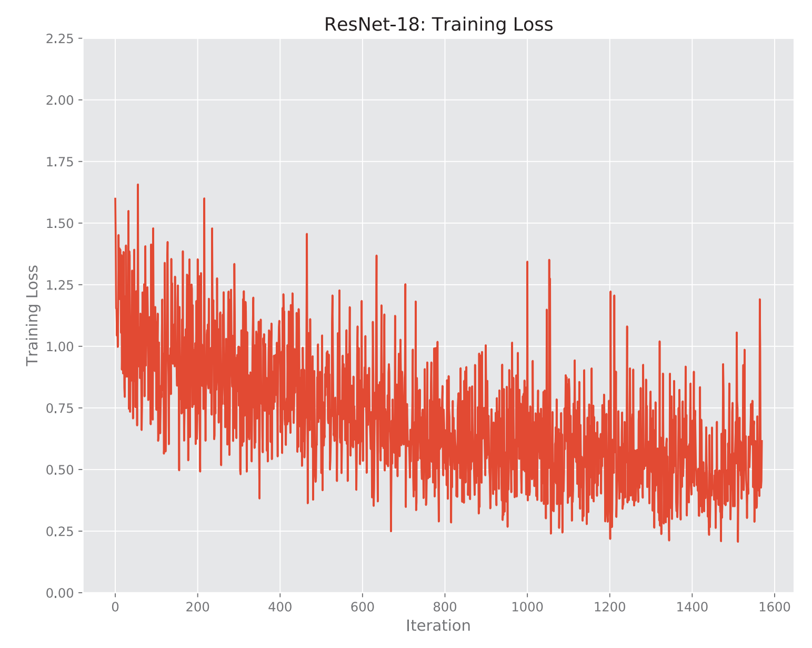
Results: HOC4

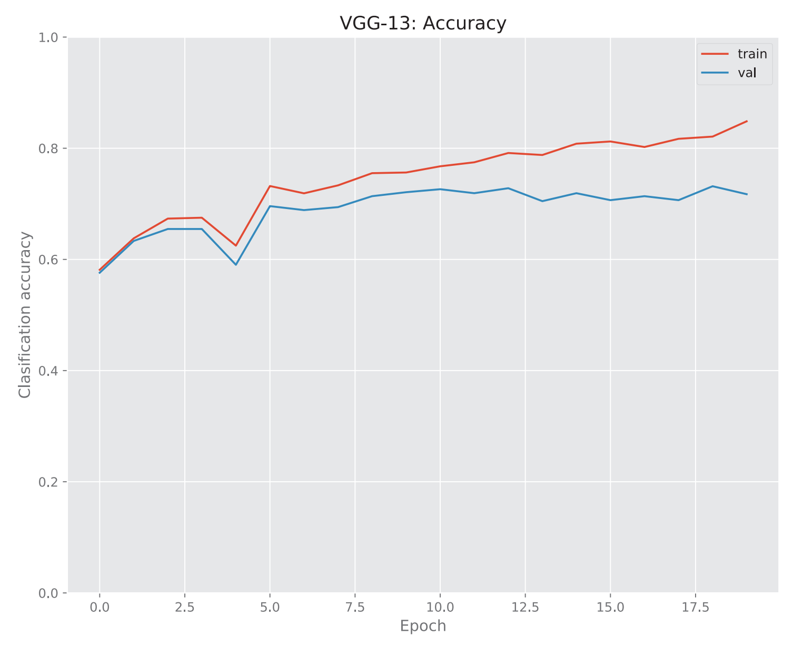
- All of the ResNet models performed well.
- The ResNet-18 model achieved the highest overall accuracy on the validation set of 0.75.
- The VGG-13 model performed best out of the VGG architectures with an accuracy of 0.73

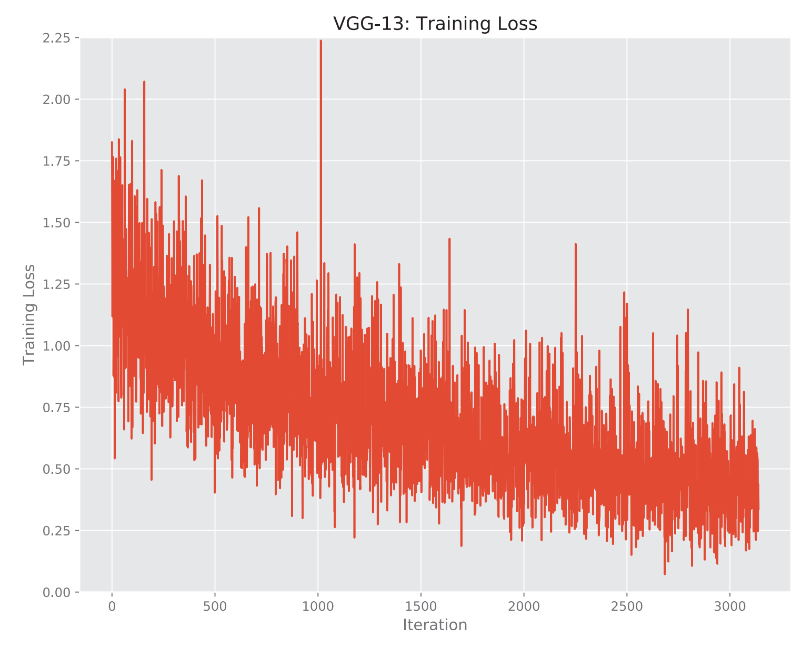
HOC4: Best Validation Accuracy

Model	Best Validation Accuracy
Baseline	0.59
Vanilla CNN	0.63
ResNet-18	0.75
ResNet-34	0.75
ResNet-50	0.74
ResNet-101	0.75
VGG-11	0.71
VGG-13	0.73
VGG-16	0.71



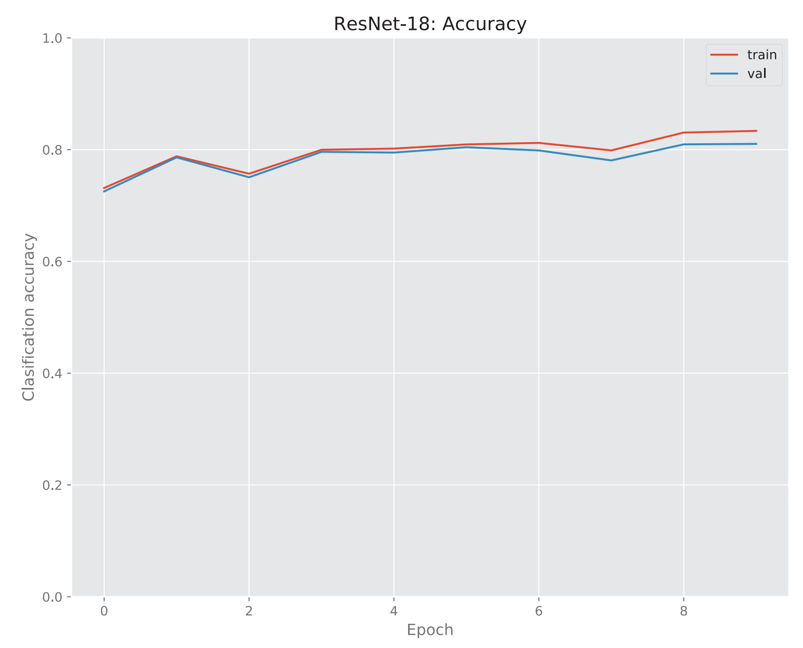


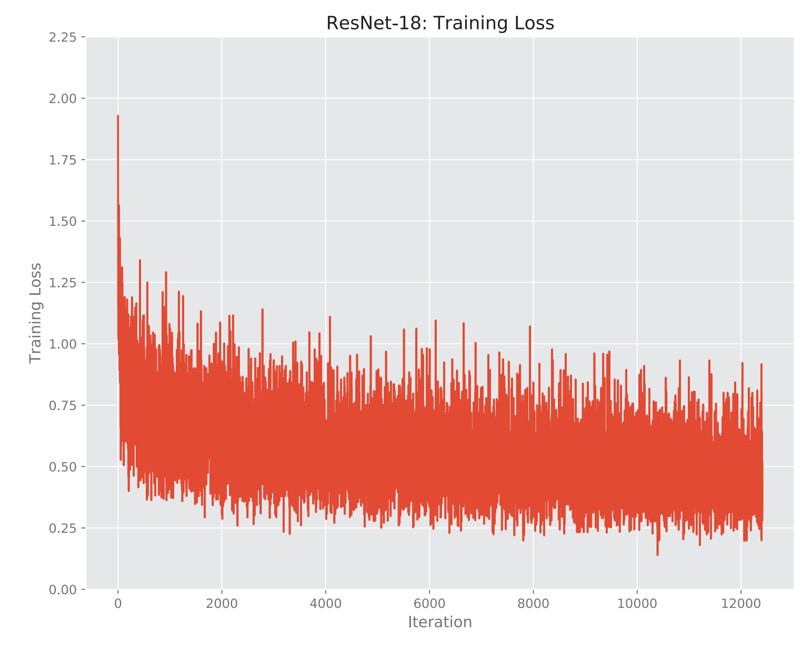




Results: HOC18

- Only top models from HOC4 chosen for HOC18
- Despite the increased complexity of HOC18, ResNet-18 achieved even higher accuracy than HOC4 (0.80). The baseline for HOC18 was 0.44.


ResNet-18: Accuracy
train
val
Classification accuracy
Epoch


ResNet-18: Training Loss
Training Loss
Iteration