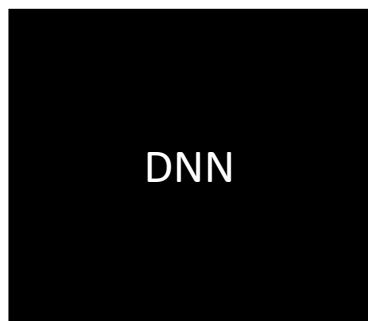


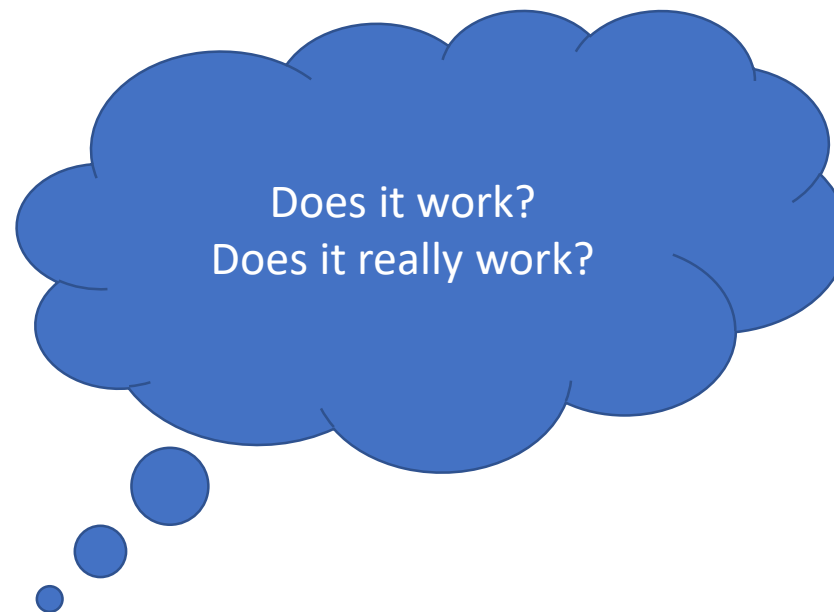
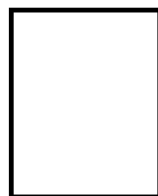
Not all Neurons are created equal:  
Towards a feature level Deep Neural Network Test  
Coverage Metric

Nils Wenzler - CSC2125: Topics in Software Engineering Winter 2019

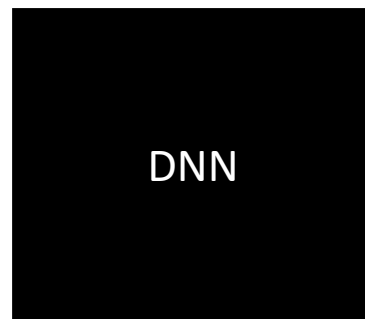
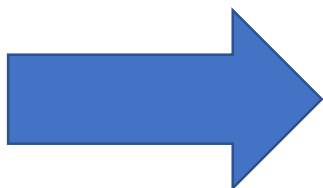
# Problem



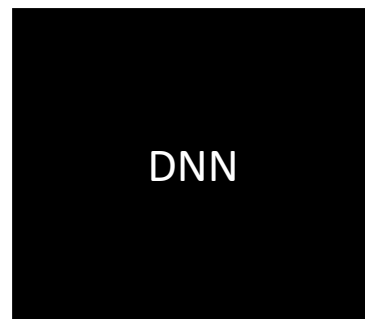
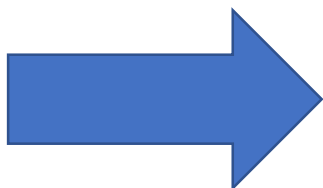
# Problem



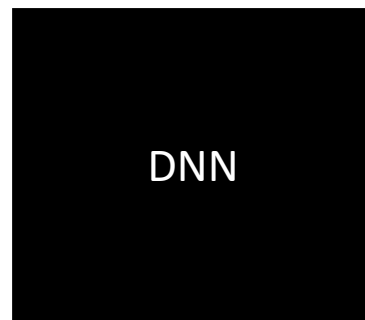
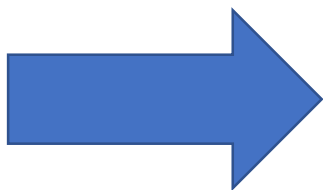
# Problem



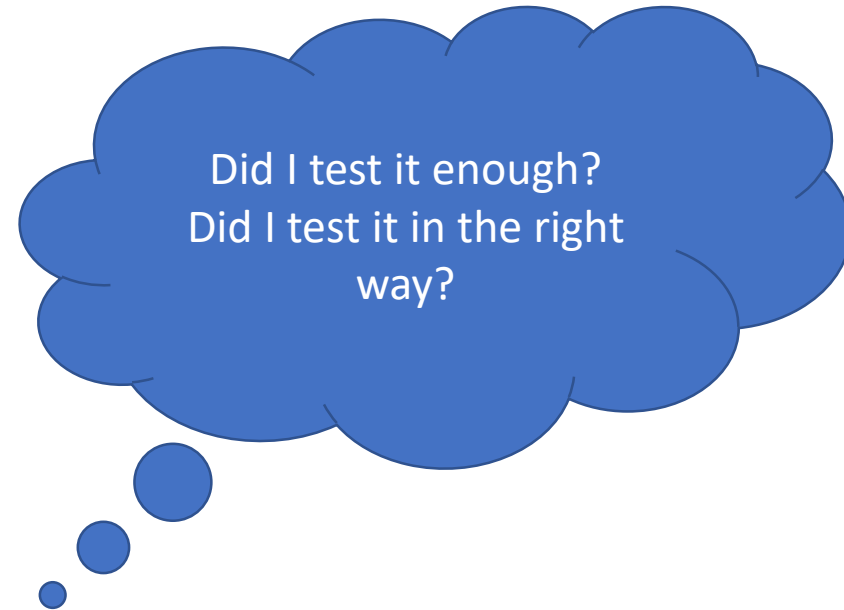
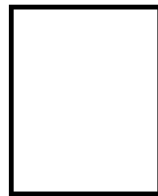
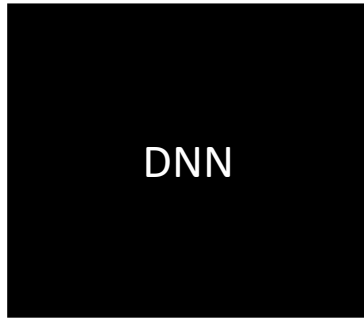
# Problem



# Problem



# Problem



# Structure

1. Problem
2. Current DNN Test Coverage Metrics
3.  $\alpha$ -Bin Coverage
4. Practical Evaluation



# General Approach

## Use a test coverage metric for

- Building test suites that
- Cover all significant behaviours of a deep neural network

Not a proof of correctness but evidence towards correctness!

### DeepXplore: Automated Whitebox Testing of Deep Learning Systems

Kexin Pei\*, Yinzhi Cao<sup>†</sup>, Junfeng Yang\*, Suman Jana\*

\*Columbia University, †Lehigh University

#### ABSTRACT

Deep learning (DL) systems are increasingly safety- and security-critical domains including cars and malware detection, where the correctness of a system's behavior for corner cases of great importance. Existing DL testing depends on manually labeled data and therefore often miss erroneous behaviors for rare inputs.

We design, implement, and evaluate DeepXplore whitebox framework for systematically testing systems. First, we introduce neuron coverage as measuring the parts of a DL system executed on manually labeled data and therefore often miss erroneous behaviors for rare inputs.

Next, we leverage multiple DL system functionalities as cross-referencing oracles to checking. Finally, we demonstrate how find DL systems that both trigger many differential achieve high neuron coverage can be rephrase optimization problem and solved efficiently by based search techniques.

DeepXplore efficiently finds thousands of near case behaviors (e.g., self-driving cars crash rails and malware masquerading as benign software) of the-art DL models with thousands of near five popular datasets including ImageNet and driving challenge data. For all tested DL models, DeepXplore generated one test input demonstrating behavior within one second while running only on its laptop. We further show that the test input DeepXplore can also be used to retrain the core model to improve the model's accuracy by up

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for this work owned by others than ACM must be honored. Credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Requested permission from permissions@acm.org.  
SIGSOFT '17, October 28, 2017, Shanghai, China  
© 2017 Association for Computing Machinery  
ACM ISBN 978-1-4503-5813-1/17/05...\$15.00  
<https://doi.org/10.1145/3132747.3132785>

#### ABSTRACT

Recent advances in Deep Neural Networks (DNNs) have led to the development of DNN-driven autonomous cars that, using sensors like camera, LIDAR, etc., can drive without any human intervention. Most major manufacturers including Tesla, GM, Ford, BMW, and Waymo/Google are working on building and testing different types of autonomous vehicles. The lawmakers of several US states including California, Texas, and New York have passed new legislation to fast-track the process of testing and deployment of autonomous vehicles on their roads.

However, despite their spectacular progress, DNNs, just like traditional software, often demonstrate incorrect or unexpected corner-case behaviors that can lead to potentially fatal collisions. Several such real-world accidents involving autonomous cars have already happened including one which resulted in a fatality. Most existing testing techniques for DNN-driven vehicles are heavily dependent on the manual collection of test data under different driving conditions which become prohibitively expensive as the number of test conditions increases.

In this paper, we design, implement, and evaluate DeepTest, a systematic testing tool for automatically detecting erroneous behaviors of DNN-driven vehicles that can potentially lead to fatal crashes. First, our tool is designed to automatically generate test cases leveraging real-world changes in driving conditions like rain, fog, lighting conditions, etc. DeepTest systematically explores different parts of the DNN logic by generating test inputs that maximize the numbers of activated neurons. DeepTest found thousands of erroneous behaviors under different realistic driving conditions (e.g., blurring, rain, fog, etc.) many of which lead to potentially fatal crashes in three top performing DNNs in the Udacity self-driving car challenge.

#### CCS CONCEPTS

• Software and its engineering → Software testing and debugging; • Security and privacy → Software and application security; • Computing methodologies → Neural networks;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for this work owned by others than ACM must be honored. Copying otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Requested permission from permissions@acm.org.  
SIGSOFT '17, May 27–June 1, 2017, Gothenburg, Sweden  
© 2017 Association for Computing Machinery  
ACM ISBN 978-1-4503-5813-1/17/05...\$15.00  
<https://doi.org/10.1145/3132747.3132785>

### DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars

Yuchi Tian  
University of Virginia  
yuchi@virginia.edu

Suman Jana  
Columbia University  
suman@cs.columbia.edu

Kexin Pei  
Columbia University  
kpei@cs.columbia.edu

Baishakhi Ray  
University of Virginia  
rayb@virginia.edu

#### KEYWORDS

deep learning, testing, self-driving cars, deep neural networks, autonomous vehicle, neuron coverage

ACM Reference Format:  
Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2017. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In *ICSE '17: ICSE '17: 49th International Conference on Software Engineering*, May 27–June 1, 2017, Gothenburg, Sweden. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3132747.3132785>

#### 1 INTRODUCTION

Significant progress in Machine Learning (ML) techniques like Deep Neural Networks (DNNs) over the last decade has enabled the development of safety-critical ML systems like autonomous cars. Several major car manufacturers including Tesla, GM, Ford, BMW, and Waymo/Google are building and actively testing these cars. Recent results show that autonomous cars have become very efficient in practice and already driven millions of miles without any human intervention [21, 36]. Twenty US states including California, Texas, and New York have recently passed legislation to enable testing and deployment of autonomous vehicles [18].

However, despite the tremendous progress, just like traditional software, DNN-based software, including the ones used for autonomous driving, often demonstrate incorrect/unexpected corner-case behaviors that can lead to dangerous consequences like a fatal collision. Several such real-world cases have already been reported (see Table 1). As Table 1 clearly shows, such crashes often happen under rare previously unseen corner cases. For example, the fatal Tesla crash resulted from a failure to detect a white truck against the bright sky. The existing mechanisms for detecting such erroneous behaviors depend heavily on manual collection of labeled test data or ad hoc, unguided simulation [11, 20] and therefore miss numerous corner cases. Since these cars adapt behavior based on their environment as measured by different sensors (e.g., camera, infrared obstacle detector, etc.), the space of possible inputs is extremely large. Thus, unguided simulations are highly unlikely to find many erroneous behaviors.

At a conceptual level, these erroneous corner-case behaviors in DNN-based software are analogous to logic bugs in traditional software. Similar to the bug detection and patching cycle in traditional software development, the erroneous behaviors of DNNs, once detected, can be fixed by adding the error-inducing inputs to the training data set and also by possibly changing the model structure/parameters. However, this is a challenging problem, as noted by large software companies like Google and Tesla that have already deployed machine learning techniques in several production-scale

arXiv:1708.08559v2 [cs.SE] 20 Mar 2018

# Current DNN Test Coverage Metrics

## DeepXplore: Automated Whitebox Testing of Deep Learning Systems

Kexin Pei\*, Yinzhi Cao<sup>1</sup>, Junfeng Yang\*, Suman Jana<sup>2</sup>  
<sup>1</sup>Columbia University, <sup>2</sup>Lehigh University

### ABSTRACT

Deep learning (DL) systems are increasingly deployed in safety- and security-critical applications such as self-driving cars and malware detection. The high variability of a system's behavior on manually labeled erroneous behavior is a major challenge. We design, implement, and evaluate DeepXplore, a whitebox testing framework for DL systems. First, we automatically generate test inputs. Next, we automatically generate test cases. Finally, we automatically check the results. Finally, DL systems that fail to achieve high test coverage are more likely to be adversarially vulnerable.

### CCS CONCEPTS

• Computing methodologies → Neural networks; • Computer systems organization → Testing and debugging

## Structural Coverage Criteria for Neural Networks Could Be Misleading

Zenan Li, Xiaoxing Ma, Chang Xu and Chun Cao  
State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China  
lizenan1995@foxmail.com, {zxm, changxin, caochun}@nju.edu.cn

**Abstract**—There is a dramatically increasing interest in the quality assurance for DNN-based systems in the software engineering community. An emerging hot topic in this direction is structural coverage criteria for testing neural networks, which are inspired by coverage metrics used in conventional software testing. In this short paper, we argue that these criteria could be misleading because of the fundamental differences between neural networks and human-written programs. Our preliminary exploration shows that (1) adversarial examples are pervasively distributed in the finely divided space defined by such coverage criteria, while available natural samples are very sparse, and as a consequence, (2) previously reported fault-detection “capabilities” conjectured from high coverage testing are more likely due to the adversary-oriented search but not the real “high” coverage.

**Keywords**—Software Testing, Neural Networks, Coverage

### 1. INTRODUCTION

Deep Neural Networks (DNNs) have demonstrated amazing performance in various tasks such as image classification and speech recognition [1]. They are also increasingly adopted in safety-critical application scenarios such as medical diagnostics [2] and self-driven cars [3]. Therefore, how to assess and improve the reliability of DNN-based systems becomes a highly relevant problem and has attracted a lot of research [4].

A particular issue is the testing of trained neural network models. In this paper, we focus on DNN-based classifiers. Different from the testing phase included in the training process that gauges the generalization of a model, here a neural network is treated as a piece of software and intentionally exercised to find potential defects when used in the real world. Depending on the application scenario, the defects hunted for can be

- **Natural inputs** that will be misclassified by the neural network. Natural inputs are those appearing in the real world, and are assumed to be distributed similarly as the training data, and the network.
- **Adversarial inputs** that can fool the network. An adversarial input, or adversarial example [5], is fabricated by, e.g., adding a well-designed perturbation to a genuine example [6].

For applications used in a friendly environment, one only needs to consider misclassified natural inputs. However in a hostile environment the threats of adversarial inputs must be taken into account. As discussed later, the distinction between natural and adversarial inputs is important.

It is challenging to test a neural network sufficiently. Different from human-written programs with unambiguous intended behavior on any legitimate input, neural networks are usually trained to provide only statistical guarantees such as accuracy and loss under the intangible IID assumption [7]. In addition, the logical interdependencies of DNNs behavior on individual examples is still an open problem [8].

Recently, inspired by the white-box testing of conventional software, a variety of structural coverage criteria has been proposed to gauge the defect-finding capability (or fault-detection capability) of DNN testing [8]. In addition to measuring the sufficiency of testing, they are also intended to guide the automated generation of test inputs and to improve DNN performance [9]. These researches are very inspiring, and some of them have been recognized with best paper awards at major academic conferences [9], [10].

However, our preliminary exploration shows that these proposed structural coverage criteria could be misleading if used without understanding their underlying principles and application contexts. This is because

- 1) The distribution of defects in human-written programs is fundamentally different from that in DNNs. As shown later, adversarial examples pervasively distributed over the finely divided space defined by given coverage criteria. On the other hand, the distribution of available natural inputs are very sparse, not to mention the rare misclassified natural inputs. That is to say, these structural coverage criteria could be too coarse for adversarial inputs and at the same time too fine for misclassified natural inputs.
- 2) Previously reported fault-detection “capabilities” of high coverage testing are more likely due to the adversary-oriented search but not the structural coverage. Our exploration also shows that the number of adversarial examples found by coverage-oriented input generation can be easily manipulated.
- 3) Our initial experiments with natural inputs denied the correlation between the number of misclassified inputs in a test set and its structural coverage on the associated neural networks.

In the rest of this paper, we will first briefly introduce structural coverage criteria for DNNs recently proposed by different authors. After that we present our analyses and experiments supporting the above arguments.

## DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars

Yuchi Tian  
University of Virginia  
yuchi@virginia.edu

Suman Jana  
Columbia University  
suman@cs.columbia.edu

Kexin Pei  
Columbia University  
kpei@cs.columbia.edu

Baishakhi Ray  
University of Virginia  
rayb@virginia.edu

### KEYWORDS

## DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems

Lei Ma<sup>1,2\*</sup>, Felix Juefei-Xu<sup>2</sup>, Fuyuan Zhang<sup>3</sup>, Jiyuan Sun<sup>4</sup>, Minhui Xue<sup>1</sup>, Bo Li<sup>5</sup>, Chunyang Chen<sup>6</sup>, Ting Su<sup>1</sup>, Li Li<sup>6</sup>, Yang Liu<sup>7</sup>, Jianjun Zhao<sup>1</sup>, and Yadong Wang<sup>1</sup>  
<sup>1</sup>Harbin Institute of Technology, China <sup>2</sup>Carnegie Mellon University, USA <sup>3</sup>Nanyang Technological University, Singapore <sup>4</sup>Yokohama National University, Japan <sup>5</sup>University of Illinois at Urbana-Champaign, USA <sup>6</sup>Monash University, Australia

### ABSTRACT

Deep learning (DL) defines a new data-driven programming paradigm that constructs the internal system logic of a crafted neuron network through a set of training data. We have seen wide adoption of DL in many safety-critical scenarios. However, a plethora of studies have shown that the state-of-the-art DL systems suffer from various vulnerabilities which can lead to severe consequences when applied to real-world applications. Currently, the testing adequacy of a DL system is usually measured by the accuracy of test data. Considering the limitation of accessible high-quality test data, good accuracy performance on test data can hardly provide confidence to the testing adequacy and generality of DL systems. Unlike traditional software systems that have clear and controllable logic and functionality, the lack of interpretability in a DL system makes system analysis and defect detection difficult, which could potentially hinder its real-world deployment. In this paper, we propose DeepGauge, a set of multi-granularity testing criteria for DL systems, which aims at rendering a multi-faceted portrayal of the testbed. The in-depth evaluation of our proposed testing criteria is demonstrated on two well-known datasets, five DL systems, and with four state-of-the-art adversarial attack techniques against DL. The potential usefulness of DeepGauge sheds light on the construction of more generic and robust DL systems.

### CCS CONCEPTS

• Software and its engineering → Software testing and debugging; • Theory of computation → Adversarial learning

### KEYWORDS

Deep learning, Software testing, Deep neural networks, Testing criteria

### ACM Reference Format:

Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems.

\*Lei Ma is the corresponding author. Email: ma@hit.edu.cn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by ACM, provided that the copies are not made for profit or commercial advantage and that copies are not made for redistribution on the Internet. Copyright © 2018, by ACM, Inc. All rights reserved. This ACM work is licensed under a Creative Commons Attribution 4.0 International License. For more information, please see <http://creativecommons.org/licenses/by/4.0/>.

## Testing Deep Neural Networks

Youcheng Sun<sup>1</sup>, Xiaowei Huang<sup>2</sup>, and Daniel Kroening<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Oxford, UK  
<sup>2</sup>Department of Computer Science, University of Liverpool, UK

**Abstract.** Deep neural networks (DNNs) have a wide range of applications.

## Concolic Testing for Deep Neural Networks \*

Youcheng Sun<sup>1</sup>, Min Wu<sup>1</sup>, Wenjie Ruan<sup>1</sup>, Xiaowei Huang<sup>2</sup>, Marta Kwiatkowska<sup>1</sup>, and Daniel Kroening<sup>1</sup>

<sup>1</sup>University of Oxford, UK  
{youcheng.sun, min.wu, wenjie.ruan}@cs.ox.ac.uk  
{marta.kwiatkowska, daniel.kroening}@cs.ox.ac.uk  
<sup>2</sup>University of Liverpool, UK  
xiaowei.huang@liverpool.ac.uk

### Abstract

Concolic testing combines program execution and symbolic analysis to explore the execution paths of a software program. This paper presents the first concolic testing approach for Deep Neural Networks (DNNs). More specifically, we formalise coverage criteria for DNNs that have been studied in the literature, and then develop a coherent method for performing concolic testing to increase test coverage. Our experimental results show the effectiveness of the concolic testing approach in both achieving high coverage and finding adversarial examples.

## 1 Introduction

Deep neural networks (DNNs) have been instrumental in solving a range of hard problems in AI, e.g., the ancient game of Go, image classification, and natural language processing. As a result, many potential applications are envisaged. However, major concerns have been raised about the suitability of this technique for safety- and security-critical systems, where faulty behaviour carries the risk of endangering human lives or financial damage. To address these concerns, a (safety or security) critical system comprising DNN-based components needs to be validated thoroughly.

The software industry relies on testing as a primary means to provide stakeholders with information about the quality of the software product or service under test [1]. So far, there have been only few attempts to test DNNs systematically [2]. These are either based on concrete execution, e.g., Monte Carlo tree search [3] or gradient-based search [4], or symbolic execution in combination

\*Felix Juefei-Xu and Ruan are supported by EPSRC Mobile Autonomy Programme Grant (EP/M019918/1). Wu is supported by the CSC-ING Oxford Scholarship.

arXiv:1708.0859v2 [cs.SE] 20 Mar 2018

arXiv:1803.07519v4 [cs.SE] 14 Aug 2018

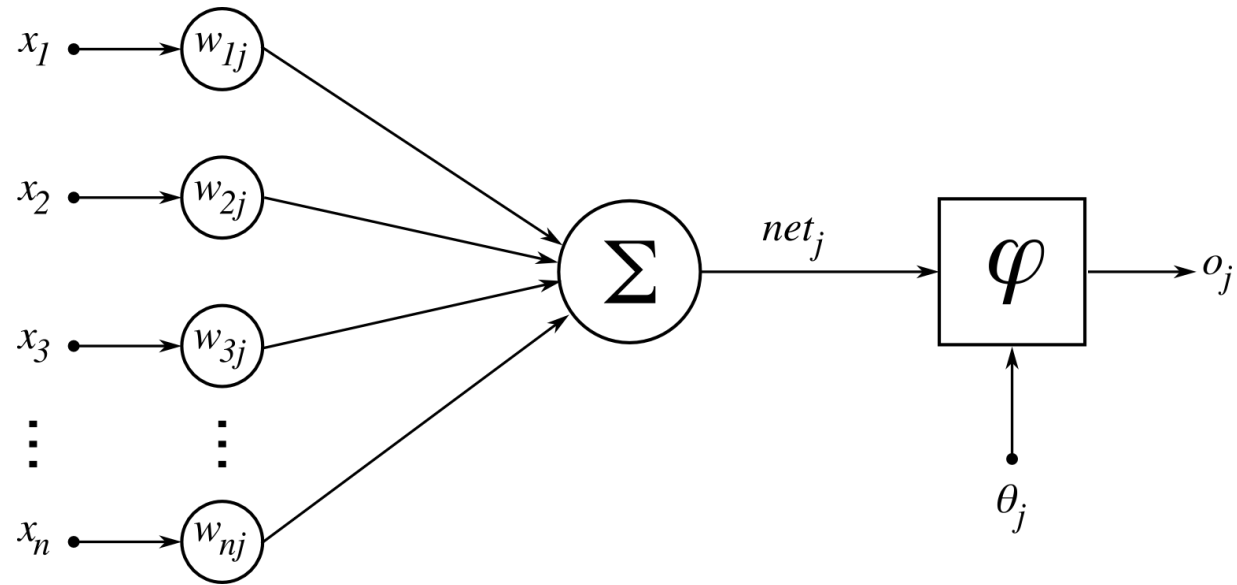
arXiv:1803.04792v3 [cs.LG] 18 Mar 2018

arXiv:1805.00089v2 [cs.LG] 4 Aug 2018

# Current DNN Test Coverage Metrics

- High research interest
- White-box testing
- Focused on single neurons

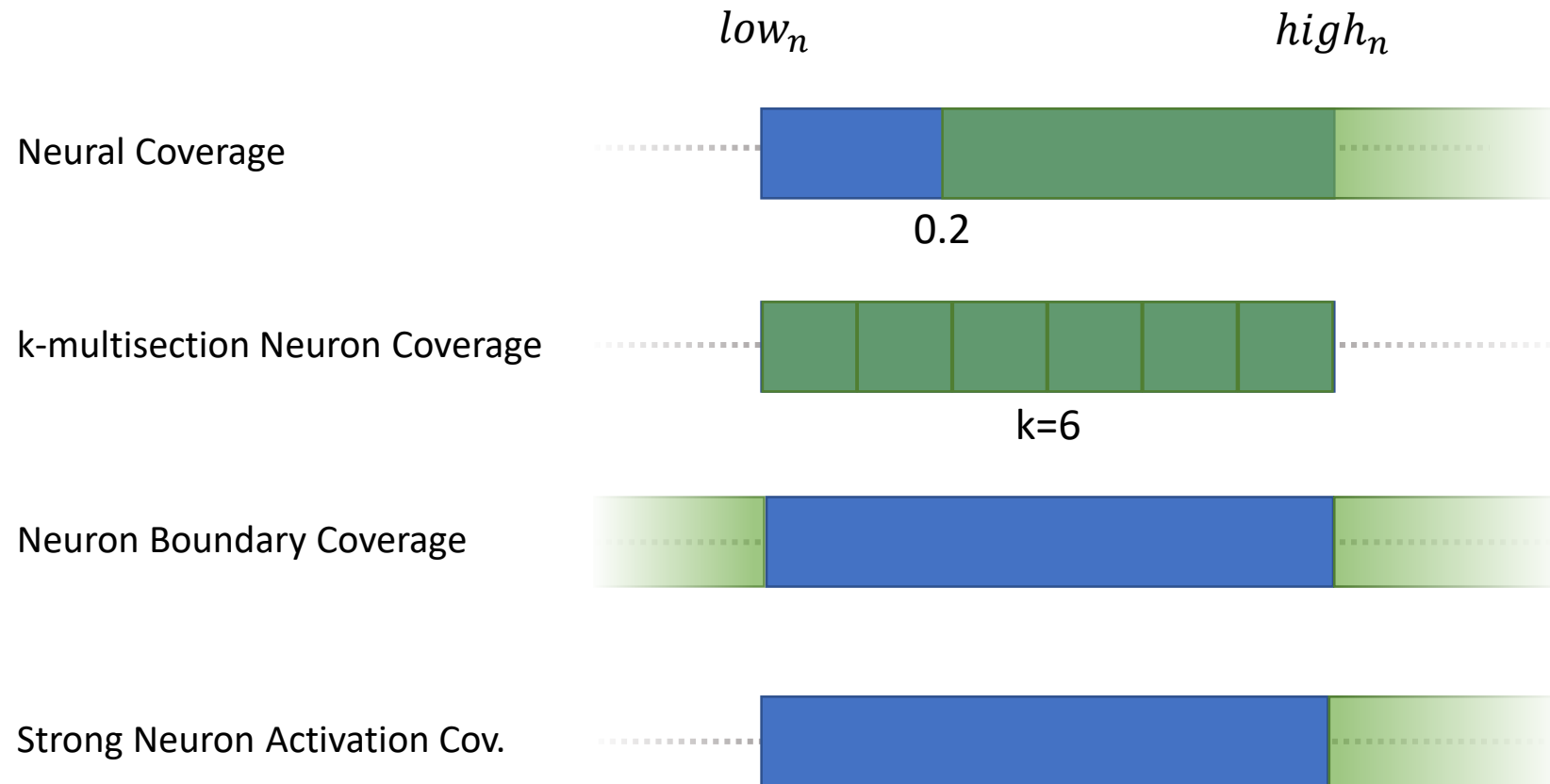
# Current DNN Test Coverage Metrics



$low_n$ : lowest output value during training

$high_n$ : highest output value during training

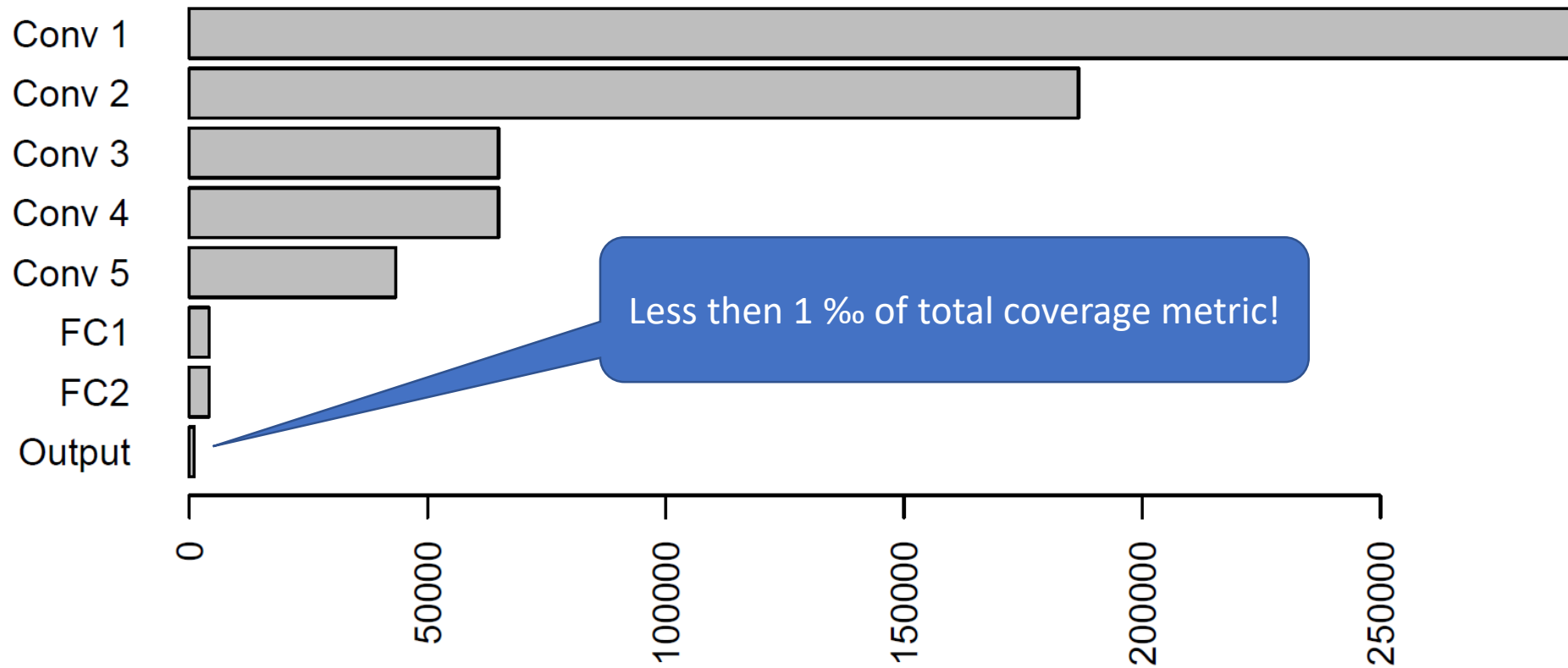
# Current DNN Test Coverage Metrics



# Structure

1. Problem
2. Current DNN Test Coverage Metrics
3.  $\alpha$ -Bin Coverage
4. Practical Evaluation

# Yet another metric?



Number of neurons per layer in AlexNet

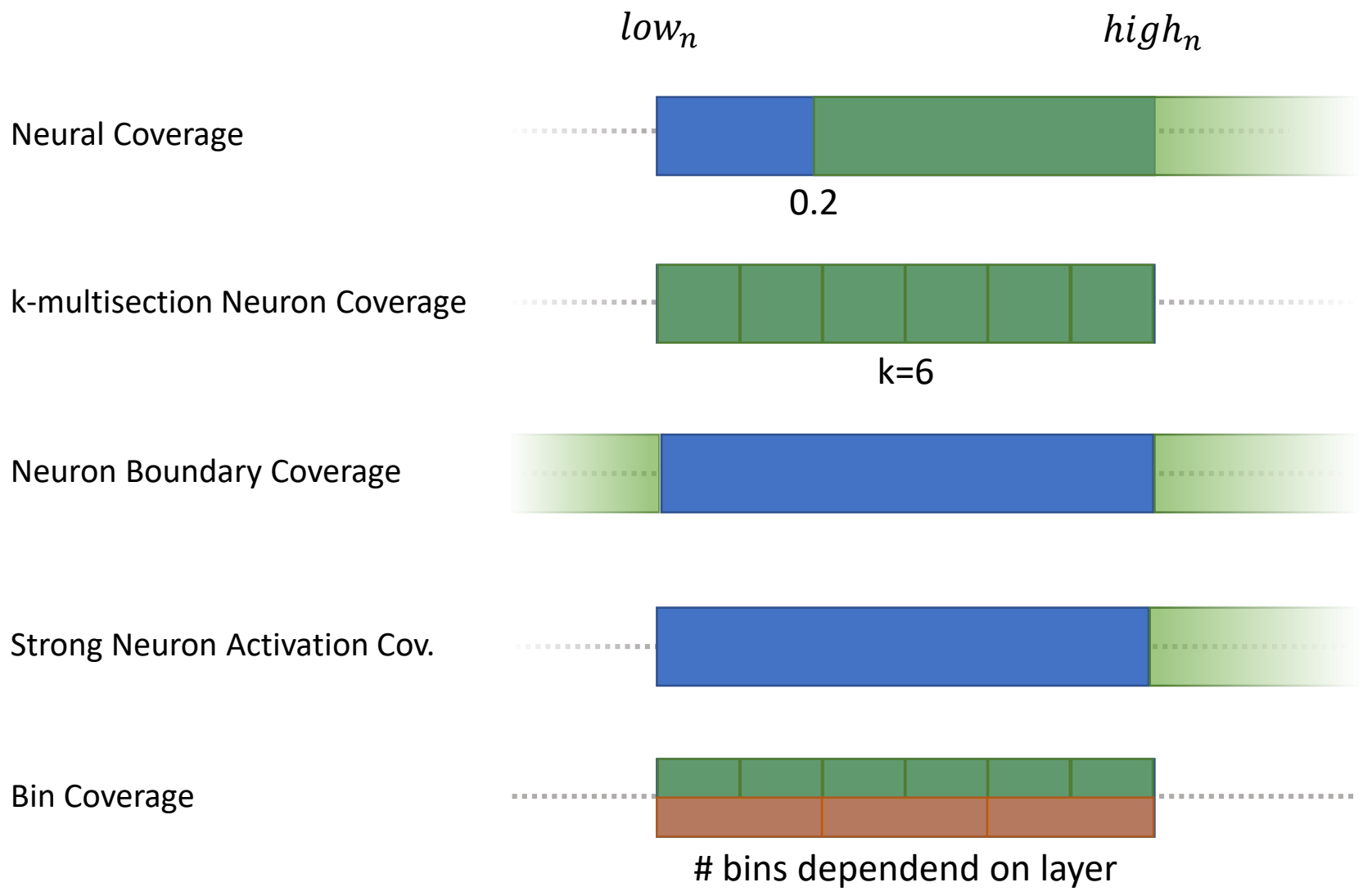
# Not all Neurons are created equal

Current metrics put equal emphasis on each neuron, but:

*Is a first layer neuron as important as an output layer neuron?*

Make use of domain specific knowledge concerning layer architectures!

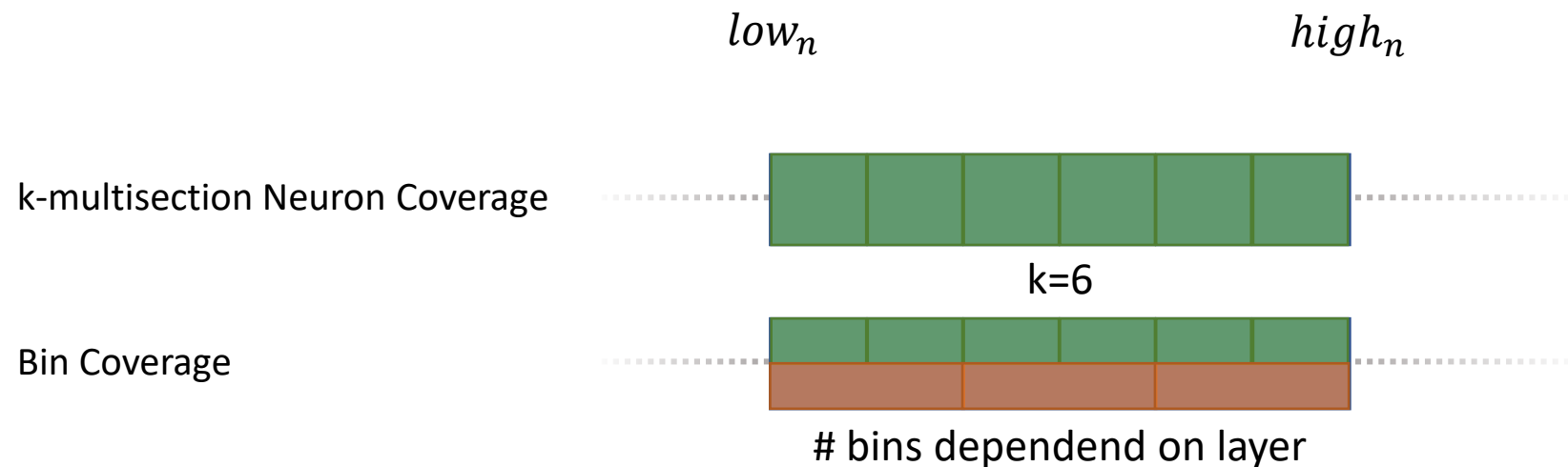




# $\alpha$ -Bin Coverage

Equally distribute so-called bins throughout layers.

Each layer contributes approximate same share to coverage metric.



# $\alpha$ -Bin Coverage

Let  $L_i$  denote the number of neurons in Layer  $i$ .

Let  $L_{max}$  be the maximum of all  $L_i$ . Let  $\alpha \in (0, \infty]$ .

The minimum number of bins per layer for  $\alpha$ -Bin Coverage is defined as:

$$Bins = L_{max} \cdot \alpha$$

The number of bins per neuron in Layer  $i$  is defined as:

$$k_i = \left\lceil \frac{Bins}{L_i} \right\rceil$$

# Structure

1. Problem
2. Current DNN Test Coverage Metrics
3.  $\alpha$ -Bin Coverage
4. Practical Evaluation

# Practical Evaluation

The main questions:

1. Can  $\alpha$ -Bin Coverage be implemented in a practically feasible way?
2. Can  $\alpha$ -Bin Coverage be optimized with a greedy search approach?
3. How does  $\alpha$ -Bin Coverage relate to other DNN coverage metrics?
4. Can  $\alpha$ -Bin Coverage be used to find wrong behaviours?

# Practical Evaluation

The main questions:

1. Can  $\alpha$ -Bin Coverage be implemented in a practically feasible way?
2. Can  $\alpha$ -Bin Coverage be optimized with a greedy search approach?
3. How does  $\alpha$ -Bin Coverage relate to other DNN coverage metrics?
4. Can  $\alpha$ -Bin Coverage be used to find wrong behaviours?

# Practically feasible?

## Test setup (1/2):

- 10 layer DNN inspired by Nvidia End to End approach using ReLu
- Trained on 45,500 publicly available labeled images
- Implemented in Python using Tensorflow



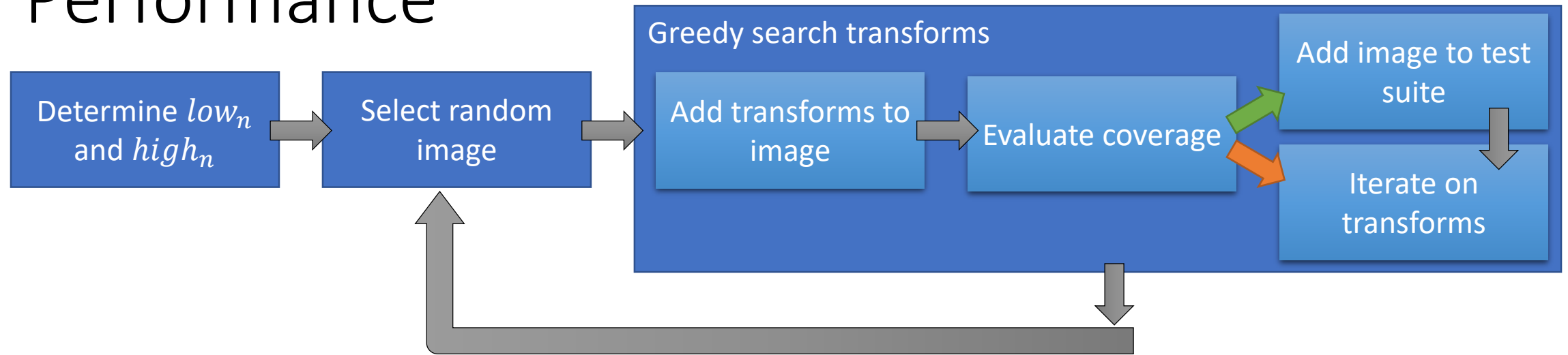
# Practically feasible?

Test setup (2/2):

- Created greedy optimizer that uses image transforms to optimize coverage metric
- Compare behaviour of  $\alpha$ -Bin Coverage & Neuron Coverage



# Performance



Determining  $low_n$  and  $high_n$  only needs to be done once and can be approximated through random sampling.

Calculating  $\alpha$ -Bin Coverage incrementally:  
constant time (dependend on network size).

# Greedy search: Transforms

Transformations: Translation, Brightness, Contrast, Blur

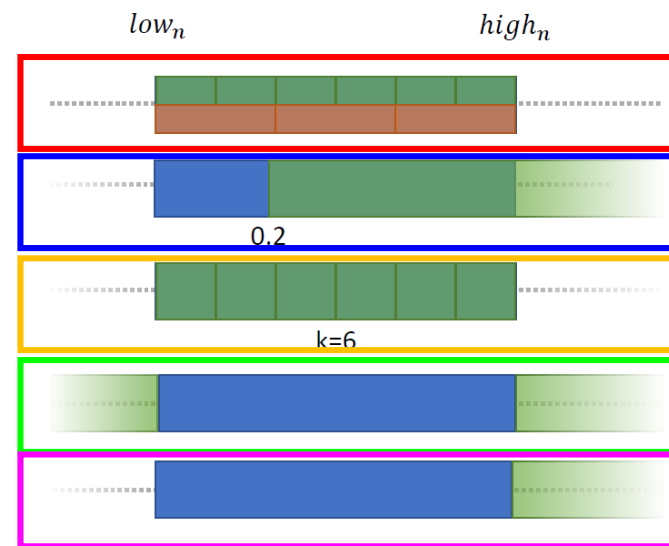
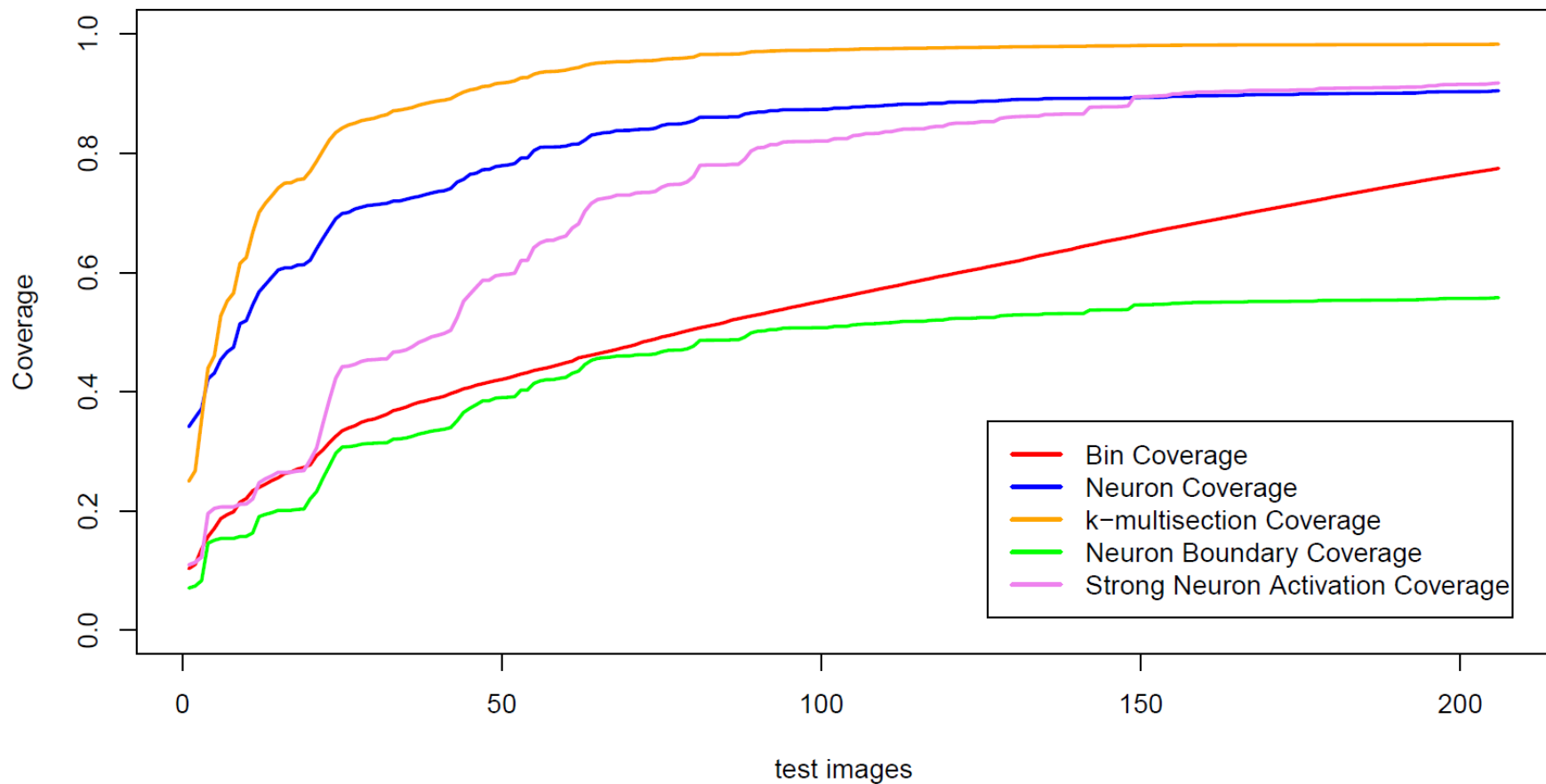


# Practical Evaluation

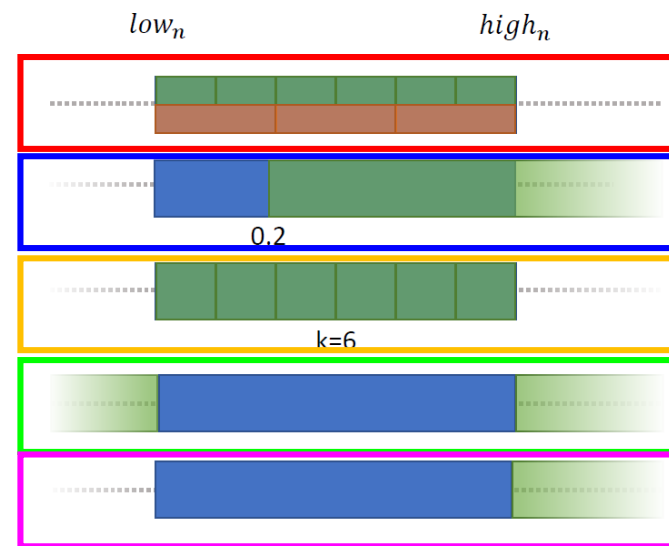
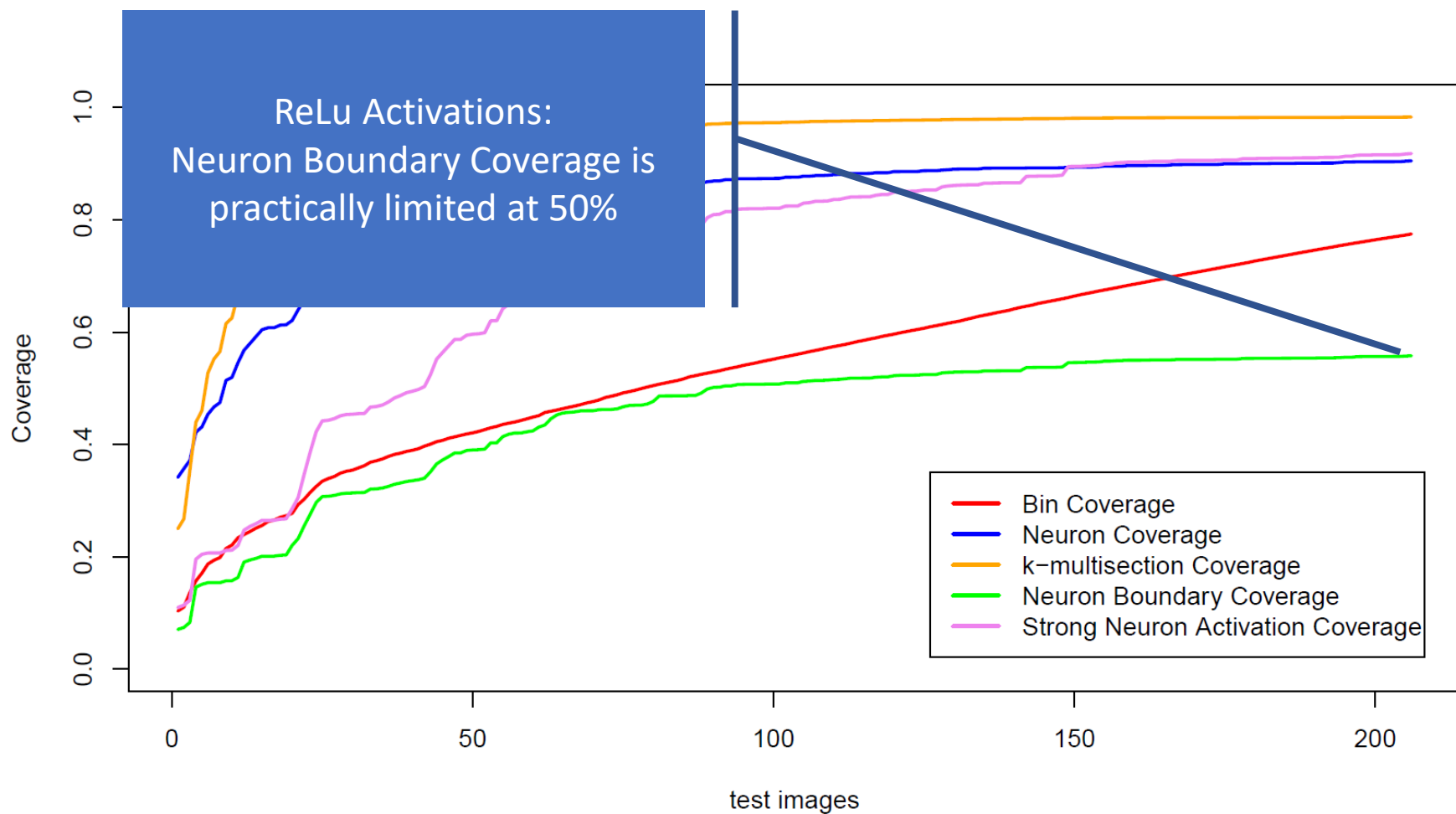
The main questions:

1. Can  $\alpha$ -Bin Coverage be implemented in a practically feasible way?
2. Can  $\alpha$ -Bin Coverage be optimized with a greedy search approach?
3. How does  $\alpha$ -Bin Coverage relate to other DNN coverage metrics?
4. Can  $\alpha$ -Bin Coverage be used to find wrong behaviours?

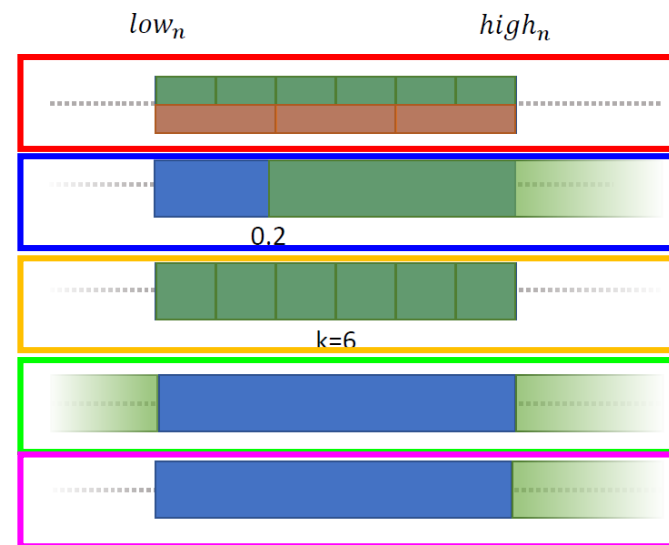
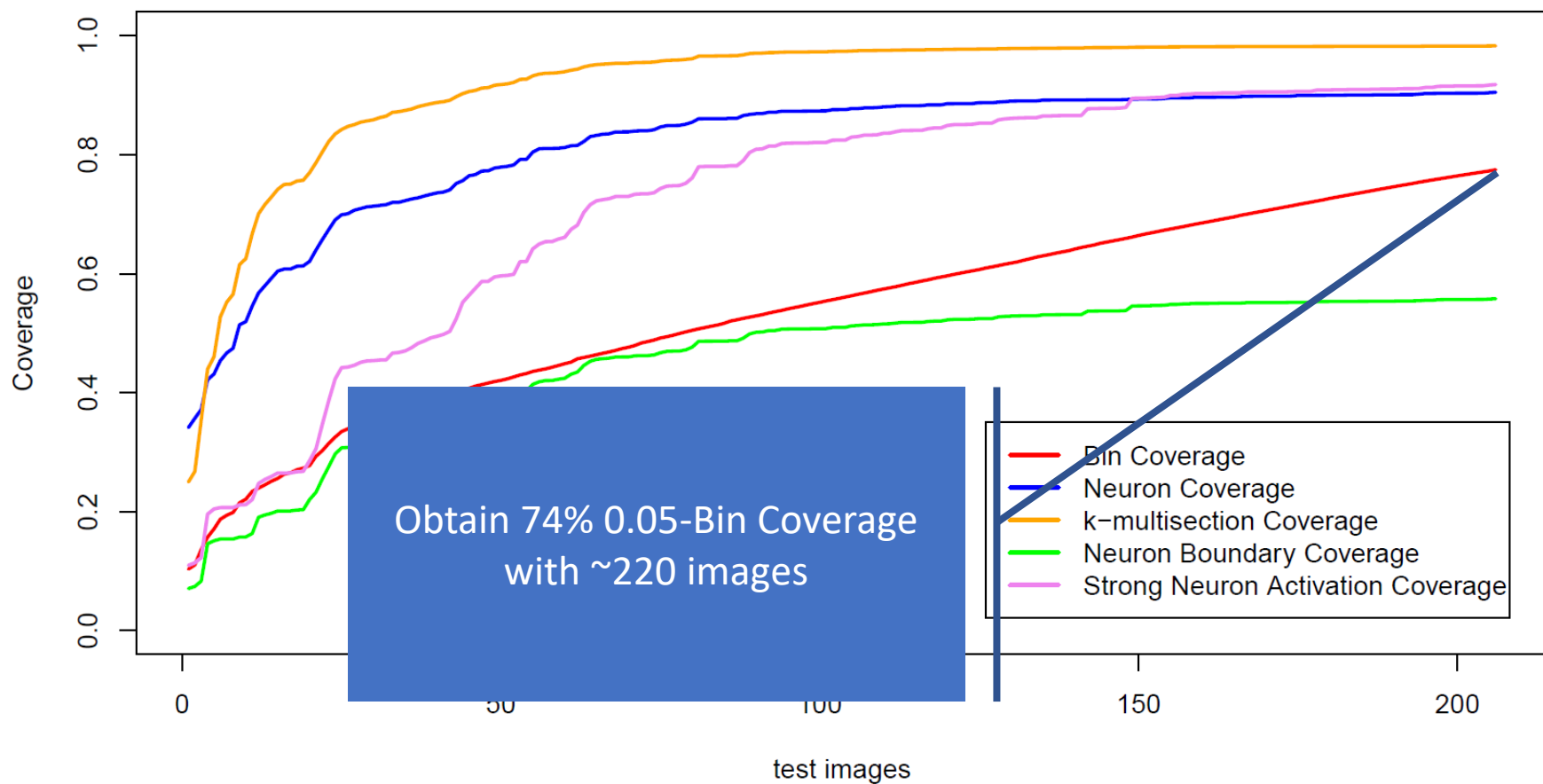
# Greedy Optimization: Bin Coverage



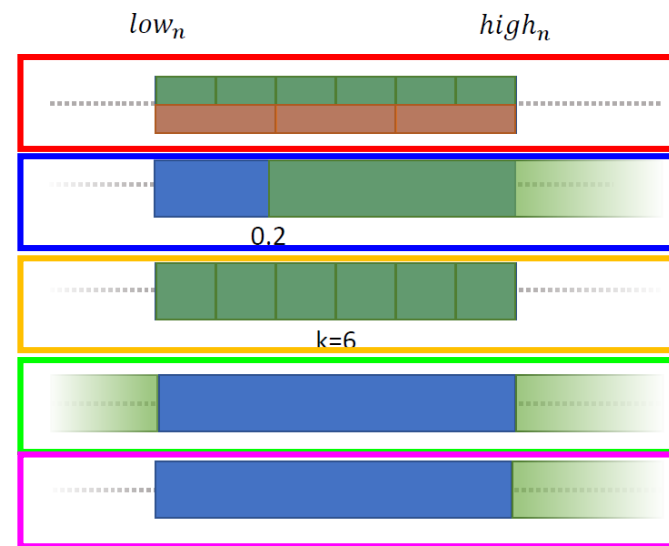
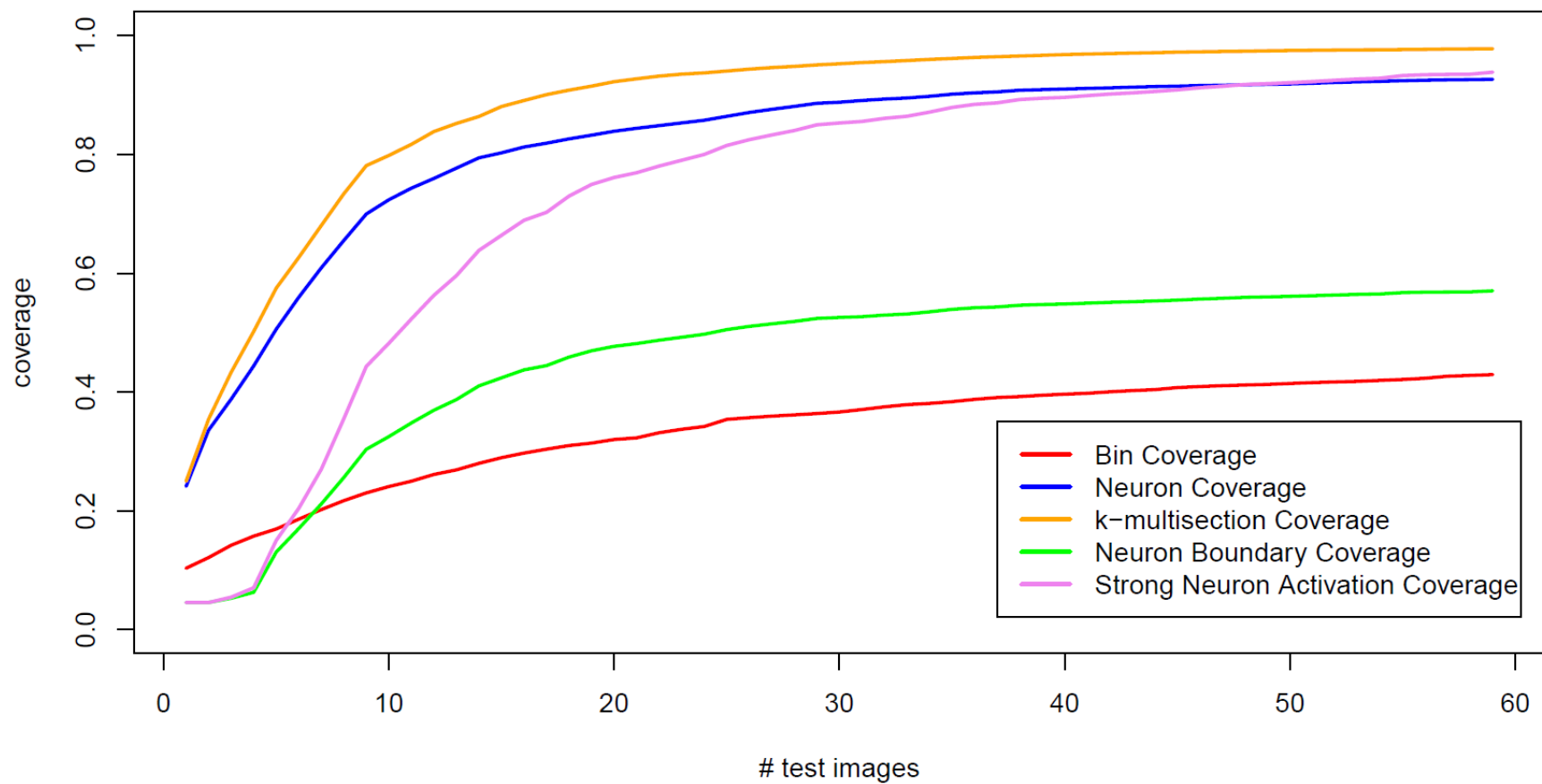
# Greedy Optimization: Bin Coverage



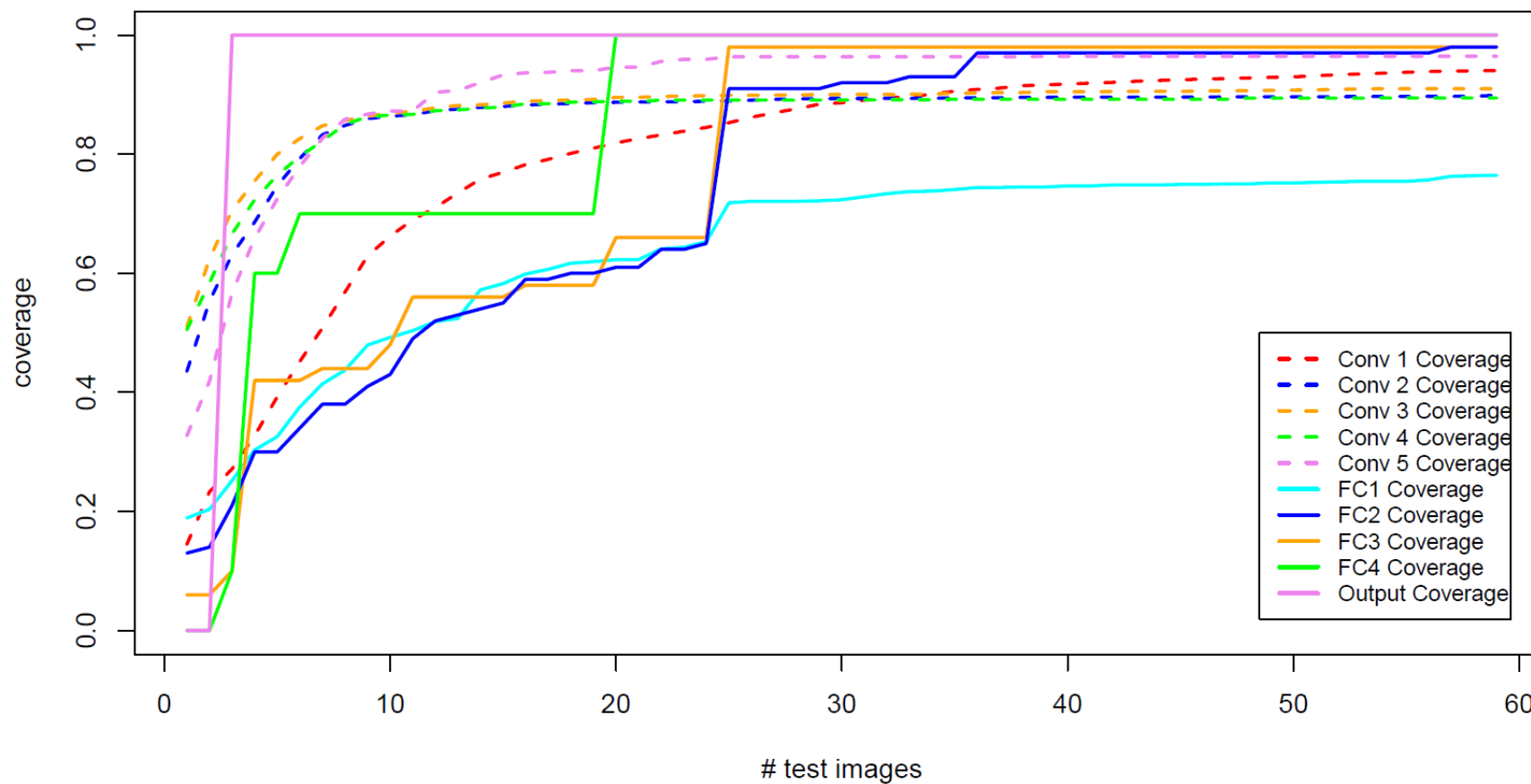
# Greedy Optimization: Bin Coverage



# Greedy Optimization: Neuron Coverage

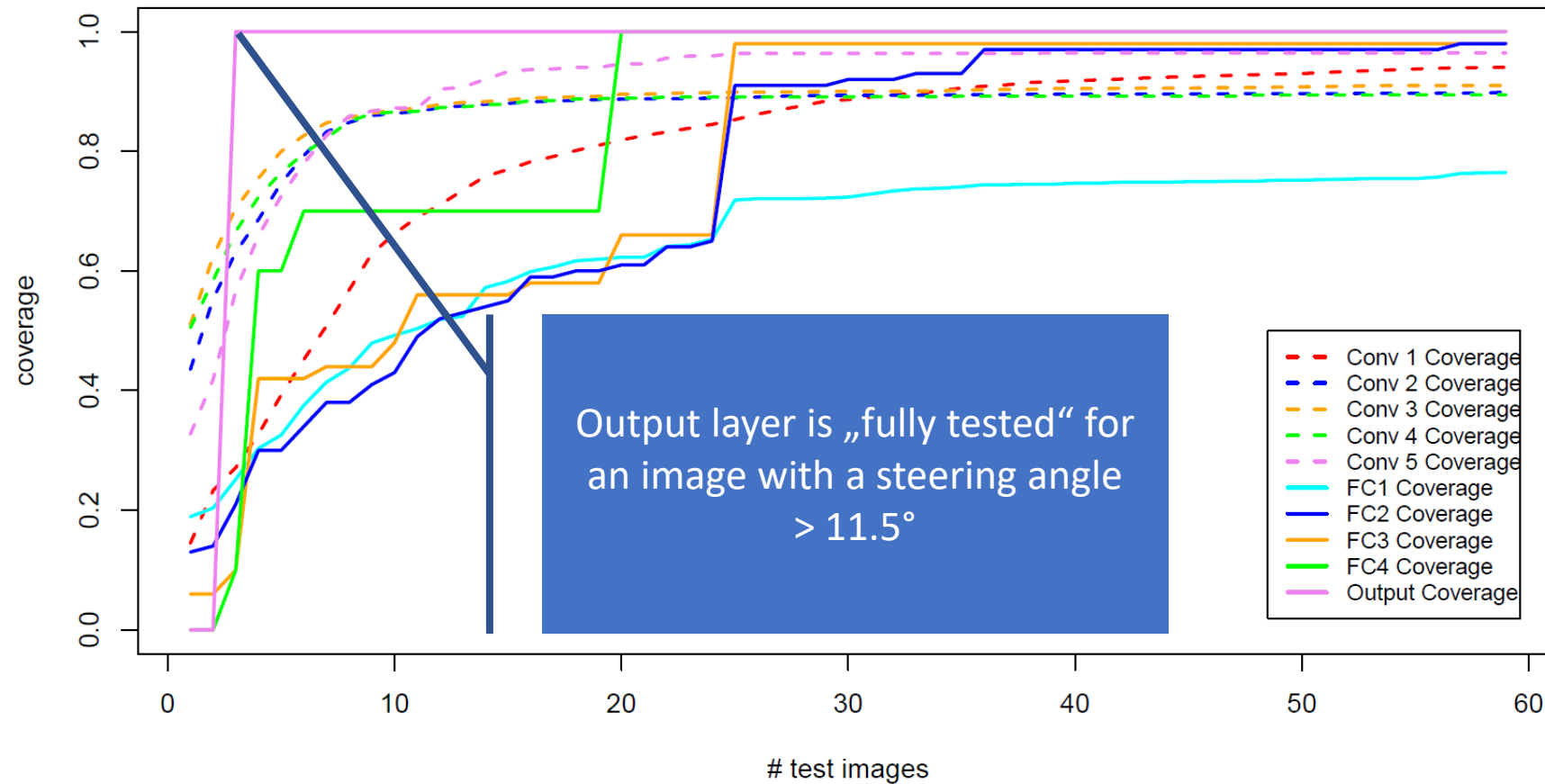


# Neuron Coverage Optimization: Layer View

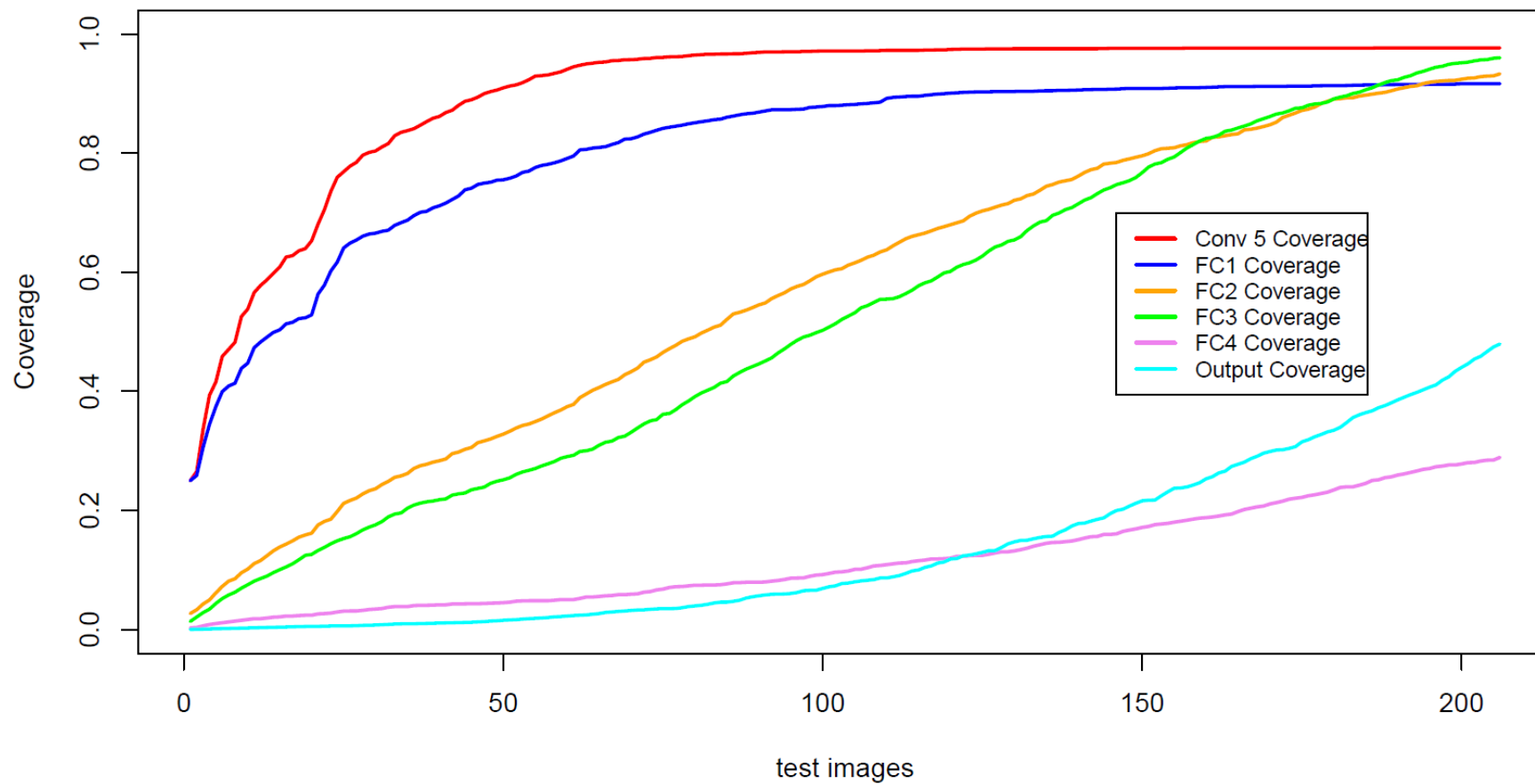




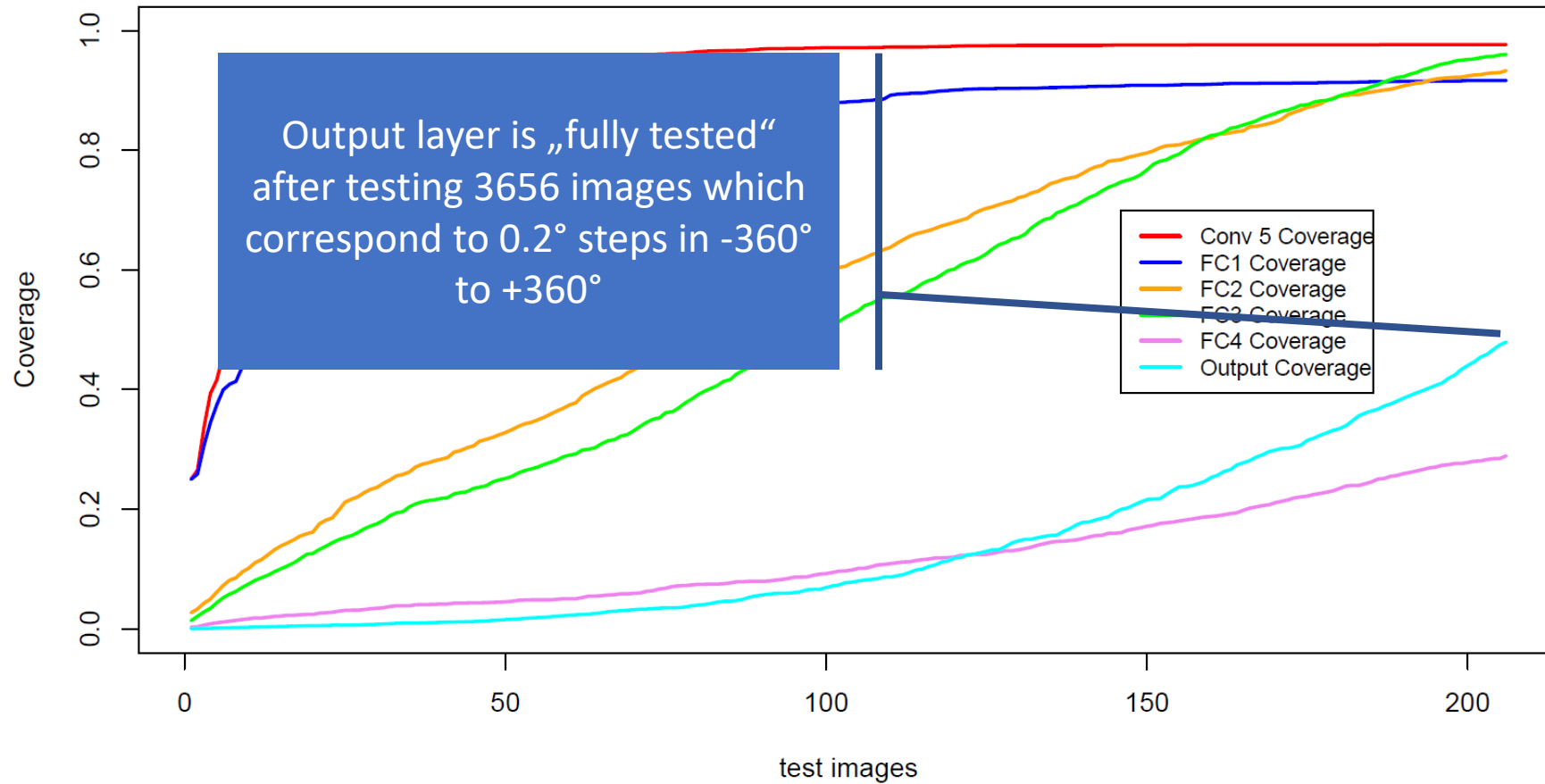
# Neuron Coverage Optimization: Layer View



# Bin Coverage Optimization: Layer View



# Bin Coverage Optimization Layer View



# Practical Evaluation

The main questions:

1. Can  $\alpha$ -Bin Coverage be implemented in a practically feasible way?
2. Can  $\alpha$ -Bin Coverage be optimized with a greedy search approach?
3. How does  $\alpha$ -Bin Coverage relate to other DNN coverage metrics?
4. Can  $\alpha$ -Bin Coverage be used to find wrong behaviours?

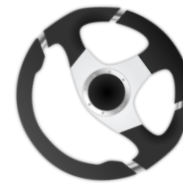
# Deviation from target labels in test suite

Coverage metric	> 20 deg	total images	ratio
Neuron Coverage	2	100	2.0%
Bin Coverage	24	247	9.7%

Example:



Transformed  
Image



Output:  
234°



Target:  
160°

# Conclusions

- Current DNN test coverage metrics deal all neurons equally
- This introduces an intrinsic focus on the neurons of low layers in modern architectures
- $\alpha$ -Bin Coverage is a practically feasible approach to equally distribute a test coverage metric over all layers
- First evidence shows that  $\alpha$ -Bin Coverage can be used for finding erroneous behaviours and creating test suites automatically

# Let's discuss!

Some points to consider:

- Only one model in evaluation
- Limited number of test runs
- Only one domain
- Why greedy search?
- What is this strange  $\alpha$  value? Why do we need it?
- How about classification tasks?

# Greedy search

Stack transformations on randomly selected images to optimize coverage metric.

Add an image to test suite if it significantly increases coverage metric

Transformations: Translation, Brightness, Contrast, Blur

