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

Nils Wenzler
nils.wenzler@mail.utoronto.ca

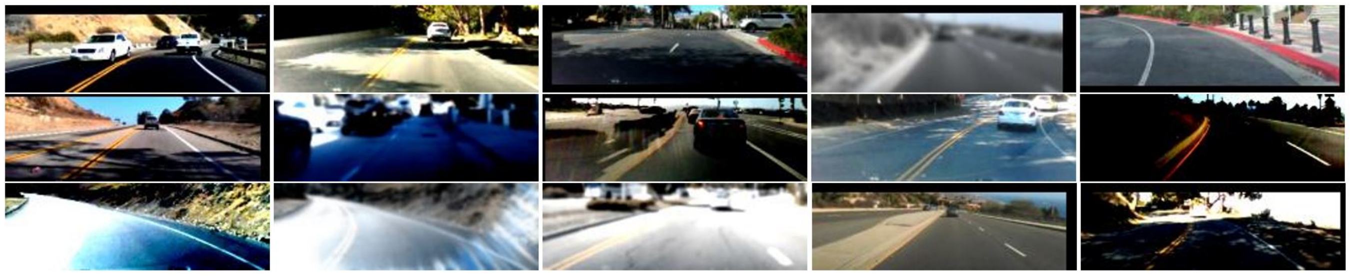

**Figure 1: A set of images optimizing Bin Coverage of a deep neural network**

## ABSTRACT

Within the last decade, deep neural networks (DNNs) have enabled computer scientists to solve tasks at an accuracy that earlier was only dreamt of. Huge advances in computer vision through DNNs power the first autonomous driving systems that nowadays are being tested on public roads. As DNNs enter such highly safety critical fields, the need for certifying the correctness of the behaviour of DNNs has risen. Since DNNs encode their decision making process in a fundamentally different way than regular human written code, the classical approaches used for testing and certifying correctness of classical code are not directly applicable to DNNs. Test coverage is one of these classical approaches to test the correctness of code. Inspired by this classical approach, researchers have thought of ways to use novel test coverage metrics that can be used for testing DNNs and to power automatic test input generation. All of the so-far proposed metrics try to port an existing classical coverage metric such as statement coverage, branch coverage and MC/DC coverage to the domain of DNNs. In this paper a new metric is proposed and empirically evaluated that is inspired by properties of modern DNNs. It is compared to existing approaches through this work and an argumentation is given, as to why such an approach might be more feasible then the so far used ones.

## CCS CONCEPTS

• **Software and its engineering → Functionality**; • **Computing methodologies →** *Computer vision problems.*

## KEYWORDS

neural networks, coverage metrics, automated test generation, bin coverage

**ACM Reference Format:**
Nils Wenzler. 2019. Not all Neurons are created equal: Towards a feature level Deep Neural Network Test Coverage Metric . In *Proceedings of* . ACM, New York, NY, USA, 8 pages. https://doi.org/

## 1 INTRODUCTION

Deep neural networks have entered many segments of every day life. Insurances, Human Resources, Healthcare and autonomous driving vehicles, all those industries are affected by huge technological breakthroughs that are powered by applying deep neural networks to their corresponding fields of expertise. As those fields have high potential impacts on the lives of people, it is important to verify their functionality. We would like to verify and guarantee that e.g. an autonomous driving vehicle is able to detect the roads, street signs and pedestrians in a reliable matter. In classical computer science, software engineers have developed a huge range of processes and approaches to build confidence in the resilience and reliability of code. Since neural networks do not encode their logic within classical code but in trained weights and biases of a network, those classical approaches can not directly be applied to neural networks. One of the most basic ways of performing classical software tests is to run tests and compute their coverage. Test cases consist of an expected behaviour given a certain program execution. A possible test coverage metric might be how many of the existing statements of the code were executed at least once when running a set of test cases. For neural networks one can think of a similar metric: Given a set of test cases, how many of the available neurons were activated (had an output value larger than a threshold value) when executing a set of test inputs. Researchers so far have tried to port coverage metrics of the classical software engineering space to neural networks. Although this approach leverages the power of known concepts of metrics, it prevents the creation of metrics which are based on intrinsic properties of modern neural networks. In this paper, a new metric is proposed that tries to be not inspired by the classical approaches and makes use of the very common availability of layered neural network architectures.

Another issue with the existing approaches is that the coverage metric is directly depending on the number of neurons. Since neurons are generally not equally distributed over the different layers, this puts emphasize on the larger layers which often corresponds to the lower layers. The approach proposed by this paper focusing on equally distributing the coverage criterion throughout the layers of the neural network.

## 2 BACKGROUND

The following subsections will introduce the theoretical foundations needed to follow through the remainder of this work.

### 2.1 Deep Neural Networks

Deep neural networks (DNNs) are an approach to enable artificial intelligence through machine learning. In DNNs a hierarchical structure of interconnected nodes that are organized in layers or different structures, are trained through training samples to optimize the process of solving a given problem. In a neural network, the single neurons perform simple calculations.

The capacity to perform complex decisions is a macroscopic result out of the single simple calculations that the neurons perform.

The simple calculation that a neuron performs consists out of two steps: First calculating the weighted sum of given input values and an additional bias term and secondly outputting the result of applying a non-linear function to this weighted sum term. Except for the last layer of neurons, the output value of a neuron becomes the input of one or more following neurons. The output of the last layer of neurons corresponds to the value(s) that the neural network calculated. During learning phase, only the weights of the inputs and their biases are fitted to the goal function. The basic decision making logic therefore can be considered to lye in these weights and therefore mostly in the connections between neurons.

Because the relationship between the macroscopic behaviour of a whole neural network and the function its single neurons compute, analyzing the correctness of the behaviour of a neural network is non-trivial.

Besides the trained weights and biases, there is another majorly important aspect of neural networks that influences how well the neural network will be able to solve a problem. This property is the architecture of the network. The architecture of a neural network describes how many neurons are used and how they are connected. Although several architectures exist, it has become common to use layered architectures. In a layered architecture, a single neuron uses some or all output values of the neurons of the lower layer to calculate its output value that will serve as another input to the next higher layer of the neural networks. While the last layer outputs the final values that the neural network was intended to calculate, the first value uses the input values of the data set as inputs.

A simplified understanding of layered neural networks is that as an input is processed through the layers of a neural network, the features that the neural network extracts to perform its calculation become more complex [8, 12]. While a first layer might detect features such as corners, a higher level might detect features such as the sky, trees or text. The most complex feature that the neural network “perceives” is the output of the last layer which represents the neural networks’ answer to the problem task it was given.

### 2.2 Coverage Testing

The problem space of showing that complex macroscopic software artifacts comply to the specified requirements is not new Classical programming uses fundamental and mostly trivial statements to build bigger software structures that solve complex problems. It was proven that showing that a general software artifact is correct is a non-computable problem Therefore, software engineering has thought of several ways of dealing with said problem. There are approaches which restrict the degrees of freedom of software which enable proofing the correctness of the code. Other approaches are satisfied with giving not proof but evidence that a software artifact behaves as specified. One such approach is the approach of testing software with test cases. A test case consists of an input data set to a program and an expected value that the software should return according to its specification. If the program returns the expected value, the test passes and otherwise it fails. This approach allows a software engineer to sample the problem input space and check the correctness for those samples.

It is important to have a measure of how well these test cases cover the problem space. Software engineers have therefore introduced so-called test coverage metrics. The most simple metric is statement coverage. It evaluates the fraction of all statements in the code of a software artifact that have been executed at least once when running the test cases compared to all existing statements of the code. More complex coverage metrics such as branch coverage require the test cases to evaluate every control flow relevant decision to true and false.

Such test coverage metrics are used in the industry to give evidence that a software was well tested and is therefore considered to be safe with a high probability [4].

## 3 RELATED WORK

Since Pei et al. [9] introduced the basic notion of coverage testing of deep neural networks with their first metric called neuron coverage, several follow-up works by different researchers have performed research on what other metrics could be feasible. This work can be considered to be one of these follow-up works. The new proposal of this paper is to consider metrics that do not put equal emphasis on the single neurons. Since the importance and the feature complexity varies throughout the different layers a neuron of the first layer should not be considered as important as the very last output neurons. The so-far introduced neuron coverage metrics are (see Figure 2 for graphical visualization):

**Neuron Coverage [9]** describes the percentage of neurons that did output a value greater than a chosen threshold. The general idea is that if the output is greater than this value, the neuron is considered to be activated. It passes a significant output to the next layer of the neural network and therefore is part of the generated output calculation. This was the first proposed deep neural net coverage metric. In a classical context, it would resemble statement coverage which checks

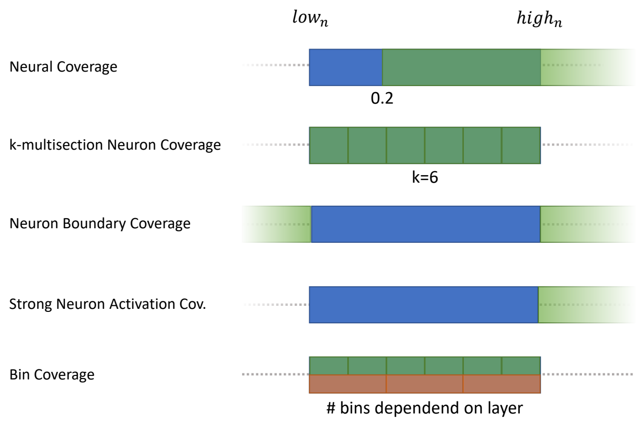
$low_n$
$high_n$
Neural Coverage
0.2
k-multisection Neuron Coverage
k=6
Neuron Boundary Coverage
Strong Neuron Activation Cov.
Bin Coverage
# bins dependend on layer

**Figure 2: Visualization of areas of a neurons' output space that different coverage metrics react to**

whether a statement has been executed at least once when executing the test cases.

**k-multisection Neuron Coverage [7]** improves on the fact that Neuron Coverage solely checks whether a neuron has been active when performing the test cases. For any neuron $n$, we denote the highest (lowest) observed output when inputting the training data set as $high_n$ ($low_n$). K-multisection Neuron Coverage splits this interval for each neuron $n$ into $k$ sections. The resulting coverage is calculated as the percentage of all sections for that at least one corresponding value was observed when inputting the test data set. In a classical context, one could compare k-multisection Neuron Coverage to branch coverage since for $k > 1$ k-multisection Neuron Coverage enforces a neuron to be in a high and a low state.

**Neuron Boundary Coverage [7]** While k-multisection Neuron Coverage focus on an equally distributed sampling of the output values throughout k equally sized sections of the single neurons, Neuron Boundary Coverage only focuses on the extreme values of high and low output. Neuron Boundary Coverage counts how many neurons have been in their upper and lower corner region when inputting the test data set. The upper corner region is defined as the region of output values in the interval of $[high_n, \infty]$. The lower corner region is defined as the region of output values in the interval of $[-\infty, low_n]$. A neuron is considered to be fully covered by test cases if output values in the upper corner region and the lower corner region have been observed.

**Strong Neuron Activation Coverage [7]** is very similar to Neuron Boundary Coverage. It corresponds to the percentage of neurons for their output values in their upper case region have been observed.

**Top-k Neuron Coverage [7]** This is a coverage metric which is observed on a layer level. For a single layer and a given input, the $k$ most active neurons are considered to be the $k$ neurons with the highest output values. A single neuron is considered to be covered by Top-$k$ Neuron Coverage if it has been for at least one test case within the $k$ most active neurons. The overall coverage corresponds to the percentage of neurons that have been covered by Top-k Neuron Coverage.

**Top-k Neuron Patterns [7]** If one looks at the combinations of the $k$ most active neurons through the different $l$ layers, one obtains an activation pattern consisting of $l \cdot k$ neurons. The resulting coverage is the percentage of observed activation patterns to the total possible activation patterns.

**Sign-Sign Coverage, Value-Sign Coverage, Sign-Value Coverage, Value-Value Coverage [10]** Sun et al. [10] propose a differently inspired set of coverage criteria. They base their coverage on the earlier mentioned notion of features, that a neural network responds to in different layers. Since it is generally unknown which features are relevant and which neurons correspond to their perception, Sun et al. define the set of features of a layer to be the set of all possible neuron sub sets of a layer.
They define the notion of a sign-change for a given feature: If for two inputs $x_1$ and $x_2$, the signs of the output values of all neurons of a feature are different in $x_2$, compared to the signs of the outputs in $x_1$, one observes a sign-change.
A value-change is similarly defined, but does not consider a sign change but a "significant" value change of the neurons. This notion of a "significant" value change has to be encoded by use of domain knowledge of the developer of the neural network.
The different coverage criteria result of different requirements to value and sign changes of features in neighbouring layers. The interested reader is referred to [10].

## 4 METHODOLOGY

All these approaches have in common that their coverage criterion is directly depending on the number of neurons. Let's consider one of the first successful deep neural network architectures, AlexNet, as an example [6]. Figure 3 shows the distribution of neurons throughout the different layers. If we consider Neuron Coverage, more than half of all neurons lie in the first two convolutional layers. The final layer which outputs the final results contributes with 1000 neurons less than 2‰of all neurons. Theoretically, a set of test inputs could reach a neuron coverage of more than 99.8% without even activating one of the final output layers. This general issue affects all of the so far presented coverage metrics.

This paper therefore proposes "Bin Coverage" as a new approach of how DNN coverage metrics can be realized while distributing the coverage criterion equally over the layers of the network.

### 4.1 Bin Coverage

Bin Coverage is an instantiation of k-multisection Neuron Coverage where k is chosen in a way such that the total number of sections (here "bins") per layer is consistent through all layers of the tested network.

The Coverage Metric has a parameter $\alpha \in (0, 1]$ that is used to scale the number of bins per layer. A high $\alpha$ value gives stronger test coverage guarantees than a lower $\alpha$ value. The total number of bins that need to be tested is proportional to the $\alpha$ value.

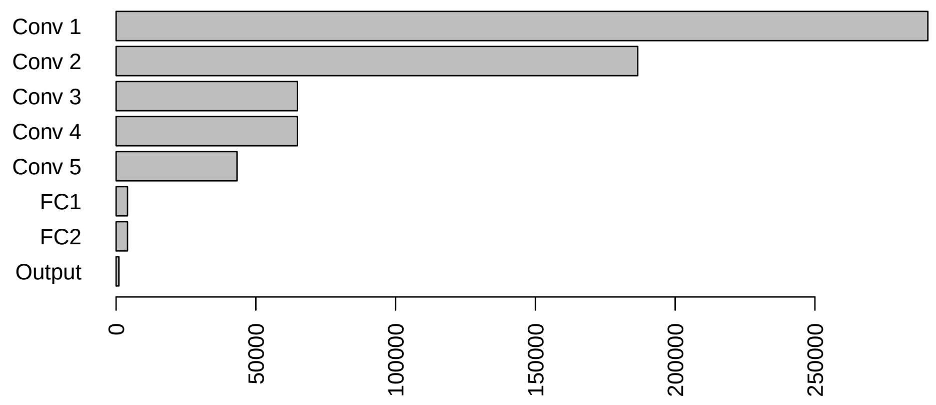

**Figure 3: Distribution of neurons in AlexNet**

Let $L_i$ denote the number of neurons in the $i$-th layer of the neural network. Let furthermore, $L_{max}$ denote the number of neurons of the largest layer:

$$L_{max} = \max(\{L_i\})$$

The minimum number of bins per layer is given through:

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

For each layer the number of bins (sections) per neuron $k_i$ is:

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

Neurons with a $k_i = 1$ are ignored because they would automatically be covered within the first test run.

The overall coverage value is the ratio of bins that were hit at least once though the test inputs.

## 4.2 Empirical Methodology

To evaluate the behaviour and performance of the proposed Bin Coverage, a lab setup was build in which a greedy search algorithm was used to optimize Bin Coverage.

One of the main goals of this empirical evaluation are to understand the relationship of Bin Coverage to other deep neural network test coverage metrics. This enables us to understand whether Bin Coverage performs differently then existing metrics. Secondly, it is of interest how Bin Coverage performs on single layers in comparison to other deep neural network metrics. The main questions are whether it is practically feasible to cover Bin Coverage on higher levels and how the layer coverage compares to a different metric (e.g. Neuron Coverage). Lastly, this empirical evaluation wants to give at least very basic evidence of whether Bin Coverage is a good metric to search for erroneous behaviour in deep neural networks.

The evaluation focuses on layered deep neural networks that perform computer vision tasks. During this optimization, other metrics such as Neuron Coverage, k-multiselection Neuron Coverage, Neuron Boundary Coverage and Strong Neuron Activation Coverage were recorded to show their relationship to the proposed Bin Coverage metric.

A special focus is put on how the single metrics behave when looking at isolated layers.

The greedy search algorithm used basic transformations to the original training images. These transformations were inspired by the transformations used in DeepTest [11].

The used transformations are depicted in Table 1.

**Table 1: Image transformations used for greedy search**

| Transform | Parameter range |
|---|---|
| Translation | $x \in [-10, 10], y \in [-8, 8]$ |
| Brightness | $b \in [-40, 40]$ |
| Contrast | $c \in [100, 154]$ |
| Gaussian Blur | $g \in [1, 2]$ |

The greedy search algorithm used for generating new input images to maximize bin coverage is given through the pseudo code in Listing 1.

**Algorithm .1: Greedy search for Bin Coverage**

```
input: set of training data images, max_tries
output: set of transformed images optimizing bin coverage
begin
  test_suite ← []
  tqueue ← []
  while coverage not sufficient
    fails = 0
    img ← select_random_image(training_images)
    while tries<max_tries
      t1 ← get_random_transform()
      t2 ← get_random_transform()
      if(tqueue.length > 0)
          t1 ← tqueue.pop()

      tqueue.append(t1, t2)
      new_img ← apply_transforms(img, tqueue)

      if(calculate_new_coverage() > old_max_coverage)
          test_suite.append(new_img)
          update_new_max_coverage()
```

```
            else
                tqueue.pop()
                tqueue.pop()
                fails ← fails + 1
        end
        return test_suite
    end
```

## 5 IMPLEMENTATION

The empirical evaluation was performed using a tensorflow [1] implementation of a deep neural network that was trained to perform steering angle prediction on single frames of video footage of a driving car [5]. Its implementation was inspired by a research project of Nvidia [2]. The data used for training the neural network is publicly available in [3].

The deep neural network used in the test setup consists out of 10 layers. The first 5 layers are convolutional layers with 72912 / 23688 / 5280 / 3840 / 1152 neurons. The following 5 layers are fully connected neurons with 1164 / 100 / 50 / 10 / 1 neurons. As with the example of AlexNet, in this setting the layers closest to the output neuron are nearly insignificant in relative size to the first layers. The output layer only corresponds to less than 0,01 fj of all neurons in the network. In this network a Neuron Coverage of 99.999% could be reached, although the final output neuron was not activated even once (following the notion of neuron activation given in Neuron Coverage) during execution of the test suite.

All neurons of the network used ReLu as an activation function.

The evaluation of the different coverage criteria was implemented in Python as well. After the initial deep neural network analysis to obtain all $low_n$ and $high_n$, the Bin Coverage can be computed in number of neurons × number of layers time when following the implementation of the greedy algorithm. It therefore can be considered to be in general technologically feasible to apply. This is a very important aspect as the number of neurons and layers might become huge and higher order polynomial times would lead to practical issues.

Because of time constraints it was sadly impossible to apply Bin Coverage to other neural networks.

## 6 RESULTS

Since one of the most interesting questions of finding a good coverage metric for deep neural networks is how a proposed metric relates to the other coverage metrics that we know, two main tests were performed. The first one did execute the greedy search on the possible transforms and the training images optimizing the total Bin Coverage of the network. During this test, the development of Neuron Coverage, k-multisection Coverage, Neuron Boundary Coverage and Strong Neuron Coverage were recorded as Bin Coverage was maximized. Figure 4 shows the results. One can see that although Bin Coverage was maximized (visible in red), k-multisection Coverage, Neuron Coverage and Strong Neuron Activation reached higher values than Bin Coverage. It seems that Bin Coverage is harder to reach and to some extend includes covering other metrics as well. This is not too surprising, especially for k-multisection Coverage (here $k = 4$) since Bin Coverage is the same approach of dividing the range of neuron outputs into different sections and checking whether these sections were hit at least once. The main difference is that Bin Coverage distributes the total number of section equally throughout the layers. One of the main claims of Neuron Coverage is that neurons are considered to be activated for an output value greater than given threshold (here 0.2 as in [11]). If Bin Coverage creates a bin which requires a value larger than the threshold (0.2), Bin Coverage implies Neuron Coverage for that neuron as well. As long as a neuron has activated at least once during training, the threshold will be part of the range $[low_n, high_n]$ and therefore this scenario can be considered at least plausible.

Neuron Boundary Coverage (here green) seems to be least covered by optimizing Bin Coverage (here red). This is a misconception which is based on a major downside of Neuron Boundary Coverage. Recall that Neuron Boundary Coverage requires two output values to fully cover a neuron. One being greater than all previously observed output values ($= high_n$) and the other being smaller than all previous observed output values ($= low_n$). Since our test network uses ReLu activation functions, the lowest possible output value is 0. This is obtained if the weighted sum of all neurons is equal or less than 0. If a neuron has at least once returned 0 as an output, it is impossible to create an input that will be lower. Since it is a likely scenario that a neuron will be inactive in at least one of the training inputs, Neuron Boundary Coverage becomes limited to a maximum coverage of 50% and becomes otherwise equivalent to Strong Neuron Activation Coverage. In Figure 4 it is visible that Neuron Boundary Coverage (here green) and Strong Neuron Activation Coverage (here violet) are strongly correlated, which gives further evidence to the mentioned explanation.

An other interesting aspect is how the total value of a coverage metric is reflected in the single coverage values of the different layers. Figure 5 shows the development of the Bin Coverage of the single layers. The plot shows only layers which have more than one bin per neuron. Since Bin Coverage distributes an equal amount of bins in the layers, for $\alpha$ values lower than 1 (here $\alpha = 0.05$) some layers will obtain 1 or less bins per neuron. Since those would be trivially satisfied, those layers and neurons are excluded from the given definition of Bin Coverage. Figure 5 shows that the lower layers are the one which obtain a high Bin Coverage first. A possible explanation would be that lower level neurons are in general easier to cover. The higher level layers need more input images to reach higher coverage. This is a direct requirement resulting out of Bin Coverage. Since every neuron only outputs one value for each input given to the network. Full coverage theoretically needs at least as many inputs as there are bins in each neuron. Since all bins are distributed equally through all layers, all bins are distributed on the one final output neuron. In the here presented test scenario, at least 3646 bins are distributed in each layer. Therefore reaching full coverage for the last output neuron would need a test suite which is at least of the size of 3646 inputs. Considering this fact, the Bin Coverage of 78% with 247 images shows that the greedy search policy was quite efficient in finding good input images.

In the first experiment, it was observed that optimizing Bin Coverage resulted in even higher values for other coverage metrics than the targeted Bin Coverage. This raises the question whether optimizing a different coverage metric than Bin Coverage could lead to a significant increase in Bin Coverage as well. Figure 6 shows the corresponding development of coverage metrics when optimizing Neuron Coverage through greedy search. Be aware that in this case only 59 images were generated since the resulting Neuron

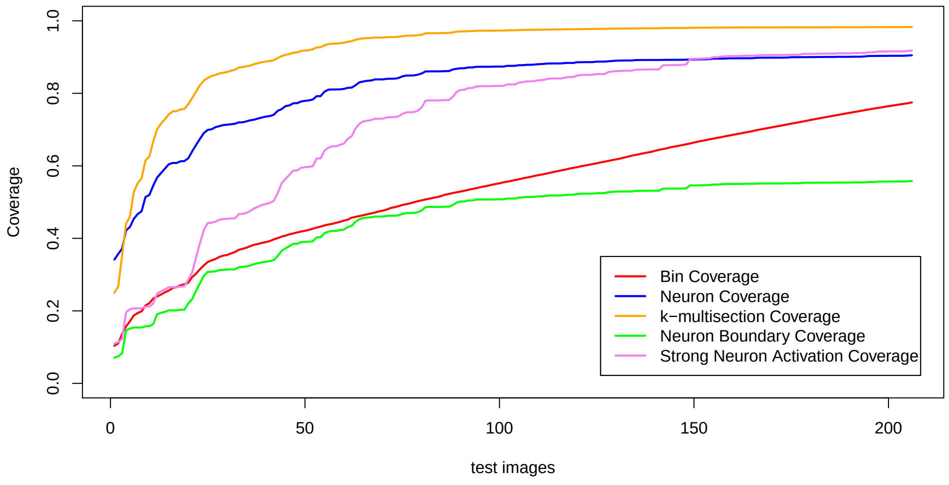

**Figure 4: Development of different coverage criteria when performing greedy search to optimize Bin Coverage**

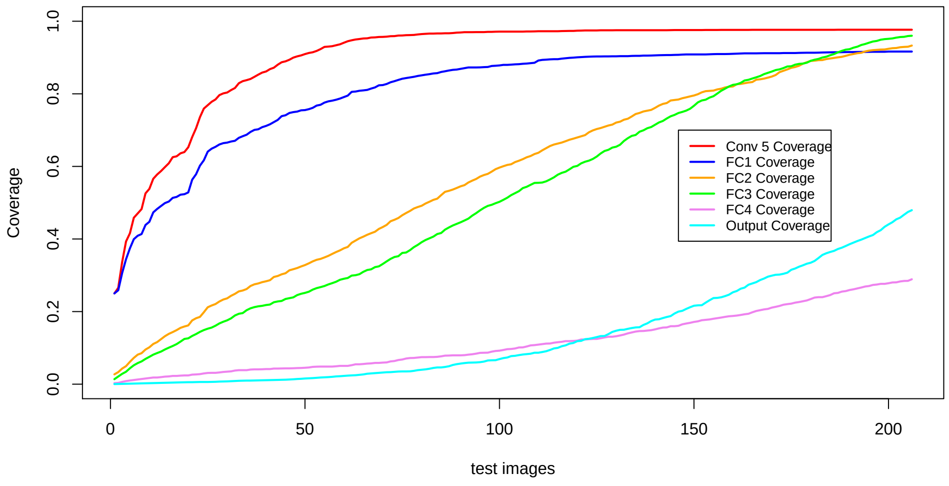

**Figure 5: Development of Bin Coverage of the different layers when performing greedy search to optimize neuron coverage**

Coverage was already close to 100%. When comparing to Figure 4 the corresponding Bin Coverage is the one at 59 images as well. At that point, the value of Bin Coverage is not significantly different, then the value obtained when optimizing for Bin Coverage directly. The main issue is that when using e.g. Neuron Coverage a tester would stop at around 59 generated images. The final resulting Bin Coverage would be 43% opposed to the 78% that we obtained after generating 247 test images through Bin Coverage driven search.

Figure 7 gives an impression considering how the total Neuron Coverage was distributed through the different layers. The output layer is "fully-tested" after the third test image was generated. To obtain this coverage, the greedy search algorithm only had to fulfill one requirement: it needed to generate a picture where the car is steering more then $0.2 \cdot \frac{180}{\pi} = 11.5\,\text{deg}$ to the right. In direct comparison: For the same "fully-tested" output layer, Bin Coverage needs a set of at least 3656 images that span the full range of the steering wheel. Furthermore, one can see that after around 28 test images, all layers except for the first fully connected layer have a Neuron Coverage significantly above 80%. One must mention that the layers containing the most neurons, the dashed lines representing the convolutional layers do not seem to oppose a challenging

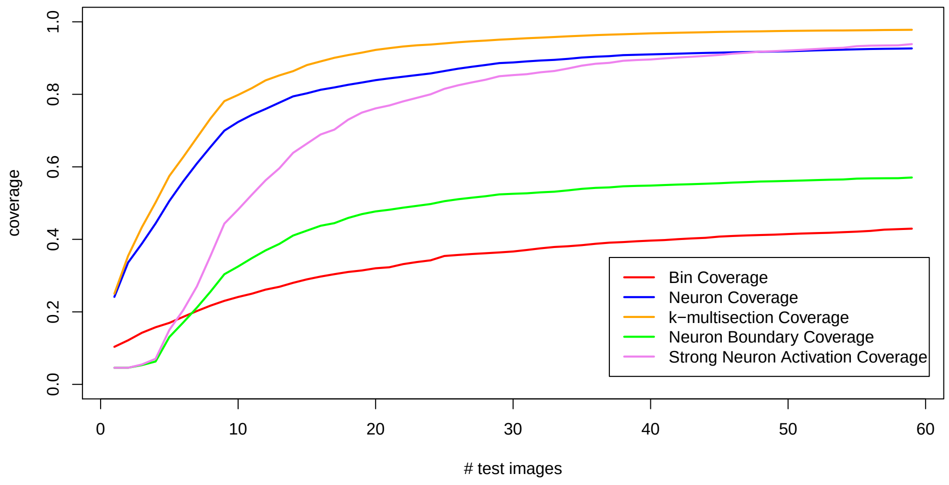

**Figure 6: Development of different coverage criteria when performing greedy search to optimize neuron coverage**

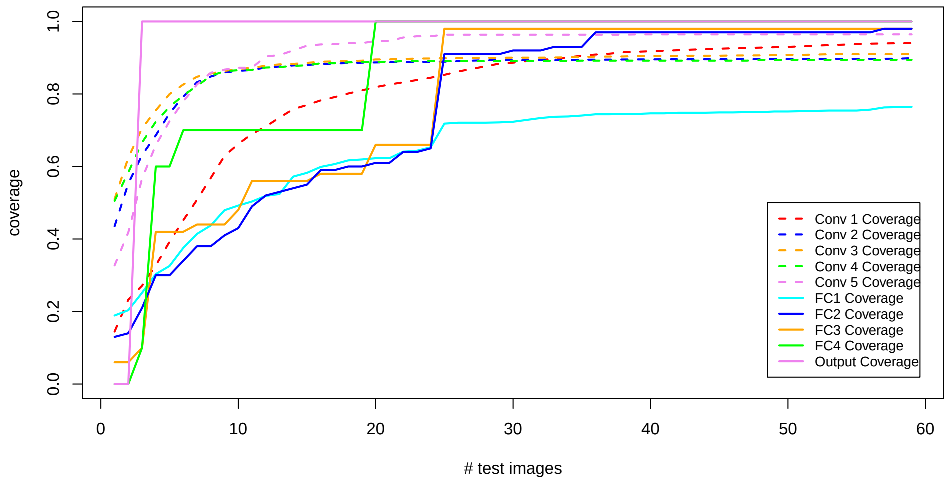

**Figure 7: Development of Neuron Coverage of the different layers when performing greedy search to optimize neuron coverage**

**Table 2: With greedy search generated images that differed more then 20**deg **from target value**

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

problem to the optimizer since they all reach very high coverage values within few images.

As a last point of evidence, this research has evaluated how many of the generated pictures were significantly diverging from their target values. For that a threshold value of 20deg was chosen. While only 2.0% of the images generated by optimizing Neural Coverage showed such divergences, 9.7% of the images generated through the optimization of Bin Coverage had such high divergences (see Table 2.

## 7 THREATS TO VALIDITY

This research is because of its short time frame very limited. The main weaknesses lie within the following aspects:

**Only one model in evaluation** The empirical evaluation only used one model to evaluate the performance of Bin Coverage. Since the implementation was performed using the very common framework tensorflow [1], the adoption process to evaluate further models should be easily possible.

**Limited sample size** The presented test runs represent a small sample size. Some further research should show how stable the performance of values obtained through the greedy search algorithm are.

**Only one domain** Deep neural networks are not only applied in the field of computer vision tasks. The performance of Bin Coverage might be limited to only this research field. This should be solved by introducing additional tests on other empirical models.

**Greedy search** Since the evaluation was heavily based on the performance of the optimization through the greedy search algorithm, it might be that some of the results reflect properties of the search algorithm opposed to properties of the coverage metrics.

$\alpha$ **value** In the empirical evaluation, only one $\alpha = 0.05$ was observed. Bin Coverage might behave differently for other values, especially for higher values.

## 8 CONCLUSIONS

In this paper, it was shown through the introduction of Bin Coverage that a deep neural network coverage metric which puts equal emphasize on the layers of the network is technologically possible. An argumentation was offered why the current notion of "All neurons are created equal" might be to the general testing disadvantage.

Through an empirical evaluation, first evidence was shown that Bin Coverage might improve on the general requirements that a test coverage metric for deep neural network imposes to reach high coverage values.

Bin Coverage leverages an intrinsic property of layered deep neural networks and pivots away from the common notion of porting existing metrics to the domain of deep neural networks.

Although this small research project sadly can not offer more then a small glimpse at the potential that Bin Coverage might have, it might be able to inspire researchers in the field to consider this variation to deep neural network test coverage metrics.

## REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[3] Sully Chen. 2018. driving-datasets. https://github.com/SullyChen/driving-datasets.

[4] ESA ECSS-E-ST. 2009. 40C Space Engineering-Software. *Noordwijk: ESA-ESTEC Requirements & Standards Division* (2009).

[5] Aditya Gupta. 2018. Self-Driving-Car-. https://github.com/cyanamous/Self-Driving-Car-.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[7] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.

[8] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The building blocks of interpretability. *Distill* 3, 3 (2018), e10.

[9] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[10] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *CoRR* abs/1803.04792 (2018). arXiv:1803.04792 http://arxiv.org/abs/1803.04792

[11] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. ACM, 303–314.

[12] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015).