Automated Testing for Self-Driving Cars

Uzair, U.S, Sipra

University of Alberta, sipra@ualberta.ca, 1498517

CCS CONCEPTS • Self Driving Vehicles • Software Testing • Neural Networks

Additional Keywords and Phrases: Coverage guided testing, Software Engineering, Automated Testing

1 INTRODUCTION

The world of deep learning has seen great strides in advancement over the past few decades in areas such as object tracking and recognition, natural language processing, robotics, and self-driving cars. With such advancements and the increasing use of deep learning in safety-critical applications, such as self-driving cars, the importance of ensuring these deep learning models possess a high degree of dependability and security has never been greater. The currently accepted norm of deep learning model performance is the statistical accuracy of test set data; however, this methodology emphasizes the t consider the semantics and behaviours of deep learning models themselves. Given the magnitude and complexity of deep learning models, measuring the dependability based solely on the output is superficial in that the foundational understanding of internal neuron processes and model semantics are not considered, leaving the dependability and quality of a deep learning model incomplete [1].

The area of self-driving cars has been of interest since 1989 when the first self-driving car was invented, called the "Automatic Land Vehicle in Neural Networks (ALVINN)", and has recently gone through monumental leaps in advancement due to modern-day high-performance processing units and access to large amounts of data. Self-driving cars are equipped with a range of sensors such as cameras, LiDAR, and IR that gather and process the surrounding environment [2]. The sensor data is used as input to a controlling deep learning model which outputs driving behaviours such as steering angle, braking, and acceleration.

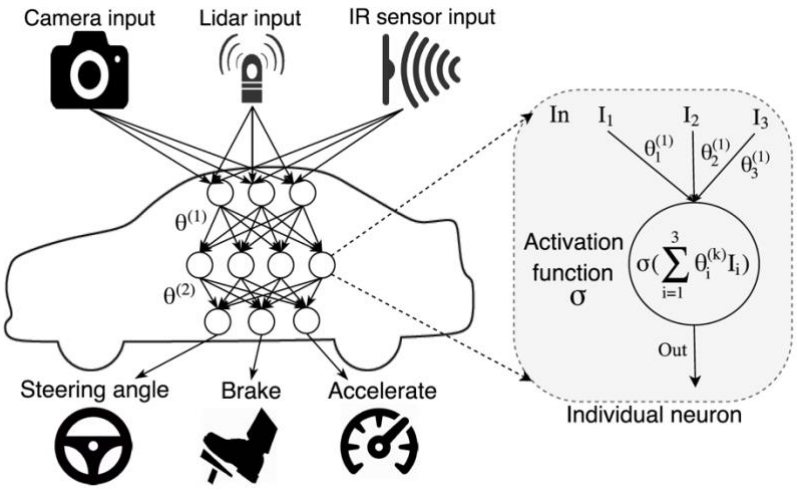

Figure 1: Autonomous vehicle system overview [3]

Recent works [4] have demonstrated state of that art learning models are susceptible to adversarial attacks, which are small perturbations on image inputs from the classified reference image. These perturbations often reside within the domain of corner cases which can result in unexpected behaviour from deep learning models, leading to fatal consequences, especially in the realm of self-driving cars. There have been several real-world examples of this behaviour such as the December 2014 crash during the Hyundai Competition [5] or the eight-vehicle crash caused by Tesla's "full self-driving" mode [6].

Inspired by the success of regular software testing, coverage metrics are intended to quantize and improve the quality of deep learning models. The prevalence of coverage metric criteria as a means of assessing the efficacy and adequacy of neural networks, such as neuron coverage (NC) [7], neuron boundary coverage (NBC), k-multisection neuron coverage (KMNC), and strong neuron coverage (SNAC), has led us to embark upon a systematic implementation of these metrics. Specifically, we propose integrating these coverage metrics with a coverage-guided fuzzing (CGF) algorithm and uniform seed sampling selection. Our approach employs metamorphic mutations on seed data for deep learning models deployed in autonomous vehicles to enhance their dependability and flexibility. Our proposal comprises a uniform distribution sampling strategy for selecting seeds (images), which will be subjected to a CGF metamorphic mutation algorithm. The structural similarity index measure (SSIM) is used to retain the core semantic authenticity within mutated seed data. We will extensively test the mutated seeds and record the outcomes against a set of coverage metrics. Through this effort, we hope to determine which coverage metric is most optimal with CGF within the context of autonomous vehicles and if the use of a uniform distribution seed selection criteria improves coverage output compared to no seed selection criteria. This effort aims to identify cases that trigger erroneous behaviour in deep learning models, thereby facilitating the optimization and re-training to yield more reliable and adaptable models.

2 PRELIMINARIES

2.1 Background

2.1.1 Object Detection and Tracking

Object detection and tracking is a core aspect of self-driving cars and therefore is briefly discussed in this section.

Object tracking is a widely researched area in computer vision with a plethora of applications such as traffic monitoring, surveillance, robotics, and autonomous vehicle tracking [8]. Object tracking can be a specialized deep learning approach where an algorithm's main goal is to track the movement of an object, which entails detecting the specified target object in the first frame of a video and tracking the target object in the remainder of the video. Object detection, a necessary process for object tracking, is where an algorithm detects an object, classifies it by producing a bounding box (Bbox) around the object, and assigns an identification for every detected unique object [9].

Object tracking is a very challenging problem in the realm of computer vision for two main reasons. The first is the degree of variability in video quality and practical factors such as illumination, video resolution, occlusions, change in target position, and rapid movements. The second is that an object tracking process is usually only given the target object in the first frame of a video, limiting the algorithm when capturing the features of the target object [10]. Despite the challenges involved in object tracking, focusing on a few key features will guide the design and development of a tracking algorithm to meet expectations [11].

Robustness: Robustness is the ability of a tracking algorithm to effectively track an object despite variations in video quality and practical factors affecting the object such as illumination, background clutter, and occlusion.

Adaptability: Adaptability is the ability of a tracking algorithm to manage the complex movement of an object by capturing and utilizing the past characteristic features of the object for future detection and tracking.

Real-time processing of information: A tracking system will typically be dealing with a video, which is just a sequence of images, which can require high amounts of computational power limiting the processing time of information, resulting in the need for highly efficient real-time processing tracking systems.

It is widely known that Convolutional Neural Networks (CNNs) are a popular choice for object-tracking tasks, where they can be used as feature extractors or binary classifiers on large image datasets. However, due to the high number of trainable parameters, updating parameters during online training can be computationally expensive, resulting in most CNN-based trackers running slower than real-time.

2.1.2 Deep Learning Overview

Deep learning is a specific subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data. All machine learning models attempt to utilize mathematical and statistical techniques to enable machines to improve on tasks with experience. A deep learning task will always require these four key components to be successful: Training/Test data, the architecture and semantics of the chosen deep learning model, loss (synonymous with cost and error) function to determine the difference between input data and predicted data, and finally, an optimization algorithm that will update the model parameters, typically a variant of gradient descent.

There is a wide range of neural network designs and architectures that build upon each other, however, the building blocks of all these networks consist of the number of layers, the number of neurons/units per layer, the number of weights that connect pairs of units from each layer together, and the function of each layer within the network listed below. The starting layer of any neural network is called the input layer, any subsequent layers are called hidden layers, and the final layer is called the output layer. The function of a neural network layer can be reduced to 4 main types listed below [12].

Fully Connected Layer: Connects every unit from one layer in a neural network to the next layer. These layers are found in all types of networks, ranging from standard multilayer perceptron (MLP) to convolutional neural networks (CNN).

Convolutional Layer: This layer is the core component of a CNN and is most used to extract feature information from images. This layer uses a filter (kernel) of a certain size that has the same dimensions as the input and will “slide” across the entire image in segments equal to the size of the filter. The filter systematically multiplies the image section it is currently on with the filter weights to extract information and check if the image section is a feature, this systematic multiplication is called convolution.

Deconvolutional Layer: This layer essentially serves the opposite purpose of the convolutional layer. The layer uses transposed convolution to upsample images to a higher resolution.

Recurrent Layer: This layer is specialized such that connections between units form a directed circle, allowing for the layer to have an “internal state” that is updated to process sequential data. This layer is the core component of a recurrent neural network (RNN) which allows them to maintain their state across iterations.

The multilayer perceptron (MLP) (Also called Feed-Forward Neural Network) can be said to be the most generic high-level neural network architecture and is the superclass of the convolutional neural network (CNN) and recurrent neural network (RNN). The “Feed-Forward” name arises from the fact that information flows one way from the input layer to

subsequent hidden layers all the way to the output layer. In simple terms, the goal of a neural network is to approximate a target function by “learning” the values of trainable parameters that result in the best approximation. The neurons are processing units that receive inputs from surrounding neurons and apply a non-linear activation function to the inputs, with the rectified linear unit ReLU being among the most popular activation functions. The width of a layer is equal to the number of neurons within that layer and the depth network is equal to the overall length of that network.

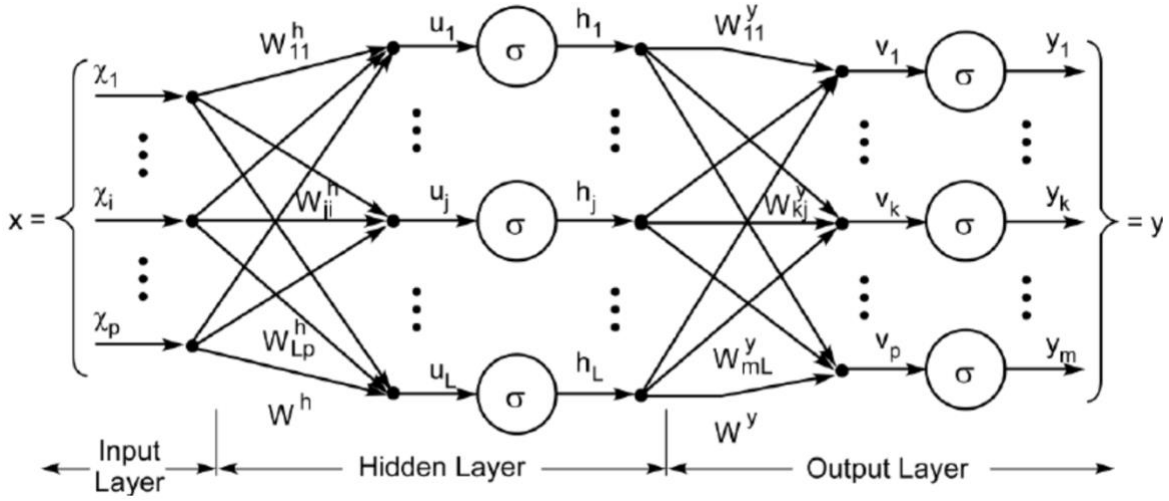

Figure 2: Multilayer Perceptron Architecture [13]

2.1.3 Back Propagation

Backpropagation serves as a versatile technique for minimizing the loss function "R" through gradient descent. This two-pass algorithm facilitates the adjustment and optimization of weights associated with a neural network. The weights are fine-tuned to minimize the sum of squared error loss for regression tasks and the cross-entropy loss function for classification tasks. The two passes encompass the "forward" and "backward" stages. For illustrative purposes, we will focus on a Single-Layer Perceptron (SLP), a neural network comprising a single hidden layer, along with the standard input and output layers. However, the underlying foundational process remains consistent across diverse neural network architectures. The loss function "R" serves as an indicator of the neural network's output accuracy in comparison to the observed or expected values. By optimizing the weights, backpropagation ensures that the network's predictions align more closely with the actual outcomes. The sum of squared error loss function for a neural network with K outputs is:

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} - f_k(x_i)\right)^2 \tag{1}$$

In the forward pass, the current weights of the network are fixed, and the predicted values are calculated using:

$$f_k(x_i) = \beta_{0k} + g_k(\sum_{m=1}^{M} \beta_{mk}\, z_{mi})\,; where\ z_{mi} = \sigma\left(\alpha_{0m} + \sum_{l=1}^{p} \alpha_{lm}\, x_{li}\right)$$

z_{mi}: *Linear combinations of observations* x_i *squashed together with the* σ *activation function*
α_{lm}: *Weights from the input layer to the hidden layer*
β_{mk}: *Weights from the hidden layer to the output layer*
m: *number of units in the hidden layer,* $1 \dots M$
k: *number of output functions,* $1 \dots K$
i: *number of observations,* $1 \dots N$
l: *number of features,* $1 \dots p$

(2)

In the backwards pass, the errors from the current model at the output and hidden layers are computed through the gradient partial derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}.$$

(3)

Where the errors for the output and hidden layers can be expressed as:

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}.$$

δ_{ki}: *Error at the output layer*
s_{mi}: *Error at the hidden layer*

(4)

In the backwards pass, first, the errors at the output layer are computed and then they are backpropagated to compute the errors at the hidden layer via:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$$
$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}$$

Where γ_r *is the learning rate.*

(5)

Once the updated weights have been computed, the entire backpropagation procedure is repeated over multiple iterations (a single sweep through the entire training set is known as a training epoch), where the weights are continuously updated, until the decrease in the loss function is marginally smaller than the previous iteration or remains unchanged.

When initiating the training of a neural network, weights are assigned random values close to zero, resulting in an almost linear model. As the weights are updated after each training epoch, the model progressively becomes more non-linear. The loss function is likely to be non-convex, possessing multiple local minima alongside a global minimum. Owing to its non-convex nature, backpropagation does not guarantee convergence to the global minimum; instead, it may settle at one of the local minima. To address this, it is advisable to train the neural network multiple times with varying random initializations for the weights and biases, enabling backpropagation to converge to different minima. The objective is not

necessarily to reach the global minimum, as it is a challenging and improbable feat. Instead, the aim is to arrive at a local minimum where the neural network's test error rate falls within an acceptable range.

2.2 Related Works

There has been a tremendous push in the research community about testing techniques and methodologies in recent times, with coverage techniques being of interest for this paper. The first proposed instance of a coverage-focused criteria was neuron coverage from DeepXplore [14]. Neuron coverage of a deep learning model can be seen as a parallel to code coverage in traditional software systems in the sense it measures what portion of the entire deep learning model is activated for a certain test input. Code coverage is the amount of code that is "activated" within a codebase for a certain input, the reason code coverage is not a competent measure for deep learning models is because, unlike traditional software that is manually written by a developer who understands the internals and semantics of the codebase, a deep learning model is "written" by being trained on data.

DeepGauge [15] proposed a set of multi-granularity testing criteria for deep learning models such as k-multisection neuron coverage, neuron boundary coverage, strong neuron activation coverage, and top-k neuron coverage. These criteria consider the understanding of deep learning models from a variety of different perspectives allowing them to expose defects within a model within the main functionality region and corner case regions. DeepHunter [16] advances this area by proposing a coverage-guided fuzzing (CGF) framework which implements metamorphic mutations on seed test inputs, generating modified tests that preserve the semantics of the original seed, and utilizes the set of coverage criterion metrics (NC, KMNC, NBC, SNAC, and TKNC) to optimize test generation and ultimately detect defects of deep learning models.

Moving onto more closely related works, DeepRoad [17] proposed an unsupervised framework to automatically generate large amounts of data using generative adversarial networks (GANs) and real-world weather condition images to test the consistency of DNN (deep neural network) based autonomous driving systems. DeepTest [3] is another work exploring the testing of DNN-driven autonomous cars by automatically generating test cases that leverage real-world changes in driving conditions and systematically partitioning DNN input-output space using neuron coverage. DeepTest employs a neuron coverage-guided greedy search technique aimed at finding generated test cases that increase coverage. The study highlights the importance of evaluating DNN-based autonomous vehicles in a manner that simulates realistic conditions, with the aim of enhancing the robustness and reliability of these systems.

While DeepTest shares some similarities with our proposal, there are crucial differences that deserve attention. One of the most significant distinctions lies in our adoption of uniform distribution sampling for seed selection, which allows for the integration of a broader spectrum of diversity into our methodology. This incorporation of diversity empowers us to generate an extensive array of test cases, essential for a more thorough assessment of deep learning models employed in autonomous vehicles.

Another notable difference is our utilization of the Structural Similarity Index Measure (SSIM) to preserve semantically meaningful mutated seed data. In contrast, DeepTest relies on the Mean Squared Error (MSE) metric for comparing seed image similarities, which is considerably less effective due to several limitations, which SSIM effectively addresses. These drawbacks include:

1. Sensitivity to noise variation: MSE is highly susceptible to changes in noise levels, leading to discrepancies between the metric's output and actual image similarities.
2. Lack of perceptual relevance: Unlike SSIM, MSE does not consider human visual system characteristics, which can result in inaccuracies between the metric's output and a human's perception of similarity.

3. Inadequate robustness: MSE is not robust to changes in image scaling, rotation, or translation that may not significantly impair the visual similarity between images.

Furthermore, our proposal aims to evaluate the efficacy of generated test cases and identify erroneous instances by employing multiple coverage metrics. This contrasts with DeepTest's exclusive reliance on neuron coverage as the sole coverage metric, which may limit the comprehensiveness of the evaluation process.

3 METHODOLOGY

To ensure the robustness and quality of deep learning-based systems used within autonomous vehicles, thorough testing is essential, just as it is for traditional software systems. However, testing metrics that are applicable to traditional software systems cannot be directly applied to deep learning-based systems due to differences in their architecture domain and fundamental paradigms. Nevertheless, the underlying concepts associated with traditional software systems can be adapted, leading to the creation of a set of coverage testing metrics that are specifically tailored for deep learning systems.

The entirety of the development and testing of our work was done through the popular deep-learning framework called PyTorch. We chose PyTorch for its great flexibility in model design and analysis and its built-in access to a plethora of open-source datasets used for model training and testing.

In this section, we detail our research methodology, including model selection and fine-tuning, dataset selection and processing, coverage metrics, and metamorphic mutation (transformation) criteria.

3.1 Dataset Selection

To progress our work, we carefully selected the cars dataset by Stanford [18] as the foundation of our experiments and testing. This dataset is composed of an impressive 16,185 high-resolution images, each depicting one of 196 different car models. The data was thoughtfully divided into two sets: 8,144 training images and 8,041 test images, ensuring that each category of vehicle was represented almost equally.

Figure 3: Sample images of the Stanford car's dataset [19]

To adapt the dataset to our specific domain, we applied several pre-processing steps to the data. Firstly, we resized the images from their original dimensions of 360x240 to 100x100. Our choice of image size was strategic, as it considerably reduced the processing time during training/testing, without compromising the essential semantics of each image.

Secondly, we introduced a few random transformations (horizontal flip and rotation) when training our model. This decision was because of the relatively small size of the dataset. Since the car's dataset is not very large, we wanted to introduce greater diversity into the data such that during training the model is exposed to different aspects of the data while slowing down overfitting. Lastly, to ensure that image features had a comparable range of values, we standardized the dataset through normalization. The process consisted of scaling the pixel values to a common scale, enhancing the model's capacity to learn from the dataset's patterns and detect features across different images.

3.2 Model Selection & Customizations

In general, there are two routes one can take when selecting a deep-learning model: build and train the model architecture from scratch or make use of publicly available pre-trained models and use them as is or apply transfer-learning for domain-specific work. Since this work's focus is not explicitly on the architectural makeup of a deep-learning model, we chose to use ResNet34 [20] and VGG16 [21], both of which are state-of-the-art pre-trained models commonly used for computer vision and deep-learning tasks.

ResNet34 is a widely used deep learning architecture that is highly effective in image classification tasks. It is a variant of the ResNet (Residual Network) family of models, which introduced the concept of residual connections to tackle the vanishing gradient problem in deep neural networks. ResNet34 has 34 layers and consists of several residual blocks, each comprising of two or more convolutional layers, a batch normalization layer, and a skip connection. The skip connection enables the model to bypass one or more layers and helps preserve important features during training. This architecture helps the model to converge faster and produce more accurate results.

VGG16 is also a popular deep-learning model introduced in 2014 by the Visual Geometry Group (VGG) at the University of Oxford. The VGG16 model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. It uses a fixed input size of 224x224 pixels and can classify images into 1,000 different categories. The convolutional layers have a small receptive field, which means they capture local patterns in the input image, and they are arranged in a sequential manner with increasing depth. The fully connected layers are responsible for making the final prediction. One of the notable features of the VGG16 model is its simplicity and uniformity, as all the convolutional layers have a 3x3 filter size and the pooling layers have a 2x2 filter size with a stride of 2. This uniformity makes the model easier to understand and fine-tune, as it is less prone to overfitting than more complex models.

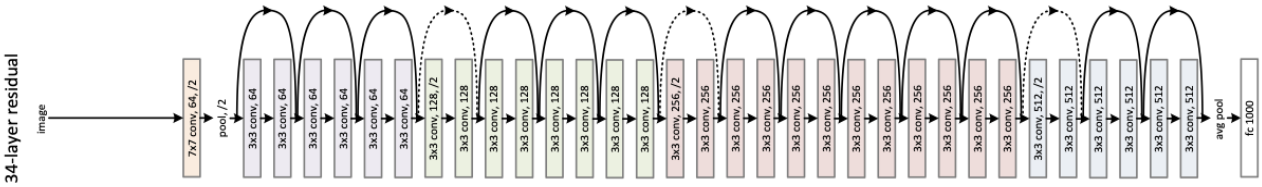

Figure 4: ResNet34 Architecture [22]

Figure 5: VGG16 Architecture [21]

3.3 Transfer Learning

Due to the limited size of the car's dataset, we used transfer learning to better the training results of our pre-trained models. Transfer learning is a technique where a pre-trained model's knowledge from a previous task can be used as a starting point for a new task. Transfer learning is very useful because it allows us to skip having to train a deep learning model from scratch, saving time and computing power. Transfer learning also introduces greater generalization because the model will already have learned useful features from a previous task and can leverage this to adapt the model to the new task more effectively.

To effectively fit the pre-trained models onto the limited data provided by the car's dataset, we utilized transfer learning, a technique that leverages the knowledge of a pre-trained model from a previous task as a starting point for a new task. Transfer learning proved to be particularly useful as it allowed us to skip the computationally expensive process of training a deep learning model from scratch. Additionally, transfer learning provides the benefit of greater generalization, as the model has already learned useful features from a previous task, which can be leveraged for the new task.

3.4 Coverage Metrics

The coverage metric implemented in this work consists of neuron coverage (NC), neuron boundary coverage (NBC), k-multisection neuron coverage (KMNC), and strong neuron activation coverage (SNAC) [23].

1. *Neuron Coverage:* NC can be defined as the neural network equivalent of code coverage for traditional software systems. A neuron is covered if its activation output is greater than a specified threshold value and is calculated via the following equation:

$$NC(T,x) = \frac{\left\{n \mid \forall x \in T, out(n,x) > t\right\}}{|N|}$$

$$n: neuron$$
$$x: input\ vector$$
$$t: threshold\ value$$
$$out(n,x): activation\ output\ of\ a\ neuron\ for\ a\ specific\ input$$
$$N: set\ of\ all\ neurons$$
$$T: set\ of\ all\ inputs$$

(6)

2. *Neuron Boundary Coverage:* NBC enhances NC by including neuron activations at the maximum/minimum boundary regions and checking if the neuron activation exceeds the maximum boundary or falls short of the minimum boundary, also called the corner-case region. A Neuron activation that exceeds the maximum boundary value is called a UpperCornerNeuron. A neuron activation that falls below the minimum boundary value is called a LowerCornerNeuron. NBC is defined as:

$$UpperCornerNeuron = \{n \in N \mid \exists x \in T: out(n,x) \in (high_n, +\infty)\}$$
$$high_n: maximum\ boundary\ value$$

$$LowerCornerNeuron = \{n \in N \mid \exists x \in T: out(n,x) \in (-\infty, low_n)\}$$
$$low_n: minimum\ boundary\ value$$

$$NBC(T) = \frac{|UpperCornerNeuron| + |LowerCornerNeuron|}{2 \times |N|}$$

(7)

3. *K-Multisection Neuron Coverage:* KMNC uniformly apportions a neuron's activation region into k sections across the range $[high_n,\ low_n]$. We define a single neuron k-multisection coverage as:

$$\frac{|\{S_i^n\ |\exists x \in T: out(n,x) \in S_i^n\}|}{k}$$

$$S_i^n: Set\ of\ activation\ values\ in\ the\ i_{th}\ section\ for\ 1 \le i \le k$$

$$k: Total\ number\ of\ sections$$

(8)

KMNC is then defined as:

$$KMNC(T,k) = \frac{\sum_{n \in N} |\{S_i^n\ |\exists x \in T: out(n,x) \in S_i^n\}|}{k \times |N|}$$

(9)

4. *Strong Neuron Activation Coverage:* SNAC is similar to NBC but only measures how many neuron activations exceed the maximum boundary region. SNAC is defined as:

$$SNAC(T) = \frac{|UpperCornerNeuron|}{|N|}$$

(10)

3.5 Mutations Technique & Structural Similarity Index Measure (SSIM)

We focuse on a unique mutation strategy, which employs a uniform distribution seed selection criterion, to generate new synthetic images with retained semantics from a set of input seeds. In this paper, we provide a detailed description of our mutation strategy and explain how we evaluate the similarity between the original seed and the mutated seed using the Structural Similarity Index Measure (SSIM).

Our mutation strategy begins by applying a set of transformations to the input seed (image) data. The selection criteria for which transformations are used are based on a uniform distribution to minimize bias during the selection phase. We run the mutations strategy k number of times on each seed, producing k new seeds that can be used for further testing. It is essential to note that we only keep mutated seeds that retain the semantics of the original seed, as it would not make sense to retain mutated seeds that are drastically altered from the original seed.

To test the semantic similarity between the original seed and the mutated seed, we choose to use the Structural Similarity Index Measure (SSIM). SSIM was first introduced in the 2004 IEEE paper, Image Quality Assessment: From Error Visibility to Structural Similarity [24]. This measure quantifies image quality based on the visible structures in the image, meaning the perceptual difference between images is measured. The SSIM value ranges between -1 and 1, where 1 means perfect similarity between the images. The use of SSIM in our work is ideal as the cars dataset and street view data seen by self-driving cars contain a vast range of information, using SSIM will allow us to retain important semantic information on mutated images.

$$SSIM(x,y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + (k_2 L)^2)}{(u_x^2 + uy^2 + (k_1 L)^2)(\sigma_x^2 + \sigma_y^2 + (k_2 L)^2)}$$

$$x, y: Images$$

$$u_x, u_y: Averages\ of\ x, y$$

$$\sigma_x, \sigma_y: Variances\ of\ x, y$$

$$\sigma_{xy}: Covariance\ of\ x, y$$

$$L: The\ dynamic\ range\ of\ pixels$$

$$k_1, k_2: Constants$$

(11)

The primary purpose of a mutation strategy is to generate a more extensive and reliable set of seed data from a smaller original set, which can be used to train pre-trained models. By comparing the coverage results of pre-trained models on both the original and mutated seed data, we can gauge the effectiveness of the mutation strategy.

Our work stands out from others in the field due to two key differentiators. Firstly, we employ a unique seed mutation strategy that utilizes a uniform distribution for selecting transformations. This approach reduces any bias that may arise during the selection process, resulting in a more diverse and representative set of mutated seeds. Secondly, we utilize the Structural Similarity Index Measure (SSIM) as our similarity measure. Unlike other similarity measures, SSIM is capable of quantifying the perceptual difference between images based on visible structures, allowing us to retain crucial semantic information in the mutated seeds.

4 EXPERIMENTAL DESIGN

4.1 ResNet34 & VGG16 Transfer Learning

To perform our experiments, we utilized the PyTorch framework and pre-trained weights of ResNet-34 and VGG-16 models. We obtained the Stanford car's dataset, which we loaded into PyTorch for training and testing. We followed standard practices for the train-test split with a batch size of 64 and enabled two parallel workers to minimize processing time. The following transformations were applied to the dataset before training:

Table 1: Pre-processing transformations

Transformation	Description	Explanation
Resize	Reshapes the image size to desired choice	Resized to 100x100 to decrease processing time and still maintain image semantics
Random Flip	Randomly flips an image along the horizontal or vertical plane	Introduces greater variability in dataset
Random Rotation	Randomly rotates an image	Introduces greater variability, we limited the rotation angle within [-15, 15]
To Tensor	Converts image to a tensor type	Pytorch works with tensors
Normalize	Normalizes an image	Ensure all image features had a comparable range of values

To fine-tune the pre-trained models on the car dataset, we replaced their final fully connected layers. Specifically, for ResNet-34, we replaced the last layer with an untrained one that contained 196 parameters. This modification allowed us to adjust the model to recognize the specific car classes in the dataset. For VGG-16, we used a more detailed replacement approach. First, we replaced the last layer with a fully connected layer containing 256 parameters and added a ReLU activation layer. Then, we included a dropout layer with a 40% setting to prevent overfitting, followed by another fully connected layer containing 196 parameters, and finally, a logistic SoftMax layer to obtain class probabilities.

During the training phase, we employed the cross-entropy loss criterion, which is a common choice for classification tasks that involve multiple classes. Cross-entropy loss computes the SoftMax activation function on the supplied predictions and then employs negative log-likelihood (NLL) to measure the difference between the predicted class probabilities and the true class labels and computes a scalar value that represents the overall loss of the network.

To optimize the network's weights and minimize the loss function, we used the standard stochastic gradient descent (SGD) algorithm. SGD algorithm computes individual adaptive learning rates for each parameter and accumulates exponentially decaying moving averages of past gradients to update the parameters' values. SGD also includes momentum, which is a technique that smooths the gradient updates and helps the optimizer navigate narrow valleys in the loss landscape. Momentum accumulates a moving average of the past gradients and uses it to scale the current gradient update. This reduces the variance of the updates and helps the optimizer converge faster. For our work, we set the momentum value to 0.9.

To prevent overfitting, we utilized a dynamic learning rate reduction technique, which reduced the learning rate by 1% after each training epoch if the accuracy of the model on the validation set started to plateau. This technique helped to stabilize the training process and improve the model's generalization performance.

We trained ResNet-34 for ten epochs and VGG-16 for twenty epochs. We observed that training VGG-16 was more challenging than training ResNet-34 because the epoch-wise accuracy increase for VGG-16 was smaller than that of ResNet-34. Nevertheless, our models achieved high accuracy on the test set, indicating their ability to generalize to unseen data.

Table 2: Details of the Stanford car’s dataset and selected DNN models

Dataset	Description	DNN Model	Accuracy
Cars	Images of cars in 196 classes	ResNet-34	89.479%
		VGG-16	86.725%

4.2 Transformation Criteria

In order to construct a robust mutation strategy, we categorized our set of transformations into two distinct groups: linear transformations and enhancement transformations. Linear transformations involve geometric modifications to the images, such as rotation, translation, scaling, and shearing. On the other hand, enhancement transformations involve pixel value manipulations, such as adjustments to color, sharpness, brightness, and contrast. To ensure that our set of transformations was representative of real-world scenarios that a self-driving car might encounter, we took great care in selecting each transformation. By including a diverse set of transformations, we can simulate a wide range of conditions, from varying weather patterns to unexpected road hazards.

Table 3: Mutation strategy transformations

Transformation	Type	Description	Explanation
Random Rotation	Linear	Augment an image by rotating it with a random angle, which is sampled from a specified range	To simulate various steering angles, a range of [-45, 45] and [-5, 5] was specified
Random Perspective		Augment an image's perspective by a random amount based on a specified distortion scale	To mimic the driving environment during turns, a distortion scale of 0.05 and 0.2 was specified
Elastic		Augment an image with elastic transformations based on a specified magnitude displacement scale	To simulate the effect of rainfall on the camera of a self-driving car, a magnitude displacement scale of 50 was specified
Color Jitter	Enhancement	Randomly change brightness, contrast, saturation, and hue of input image	To simulate a range of lighting conditions that a self-driving car might encounter during operation.
Random Erasing		Augment an image by randomly deleting pixels	Replicate obstructed vision conditions that can occur while driving
Random Adjust Sharpness		Augment image sharpness by a random amount based on a specified sharpness factor	Replicate effects of foggy driving conditions with specific sharpness factors of 0.2, 0.6, and 0.9

5 EXPERIMENTAL RESULTS

In this section, we present the experimental results of our coverage metric implementations on the ResNet-34 and VGG-16 models. Our objective was to evaluate the effectiveness of our work in capturing the adversarial robustness of deep learning models used in autonomous vehicles. We also investigated the impact of different parameters, such as the k value in the KMNC metric and the k value in our mutation strategy, on the coverage results.

To establish baseline results for comparison, we first ran our coverage metric implementations on the base pre-trained models. We used a k value of 7 for the KMNC metric as a starting point. Table 4 presents the coverage results on the base pre-trained models:

Table 4: Coverage results on base pre-trained models

Model	NBC	KMNC (K = 7)	SNAC
ResNet-34	0.34382	0.21856	0.51259
VGG-16	0.23698	0.22302	0.34765

As shown in Table 4, the NBC and SNAC metrics achieved relatively high coverage scores, while the KMNC metric had a lower coverage score.

Next, we performed experiments with various k values for the KMNC metric to observe the impact on the coverage results. Table 5 presents the KMNC results on the base pre-trained models with varying k values:

Table 5: KMNC results on base pre-trained models with varying K values

Model	K (KMNC)	KMNC
ResNet-34	3	0.36535
	5	0.25529
	10	0.18298
VGG-16	3	0.35403
	5	0.24497
	10	0.18891

Based on the results in Table 5, we observed that a lower k value for KMNC resulted in a higher coverage. Therefore, we selected a k value of 3 for KMNC in our subsequent experiments.

To further evaluate the effectiveness of our coverage metrics, we applied our mutation strategy implementation to the test dataset. We fed the test dataset multiple times with different k values to produce k mutated images for each image in the dataset. We used different k values to evaluate the impact on the coverage results. A k value of 0 indicates the original seed images were removed from the dataset, resulting in a dataset with exclusively mutated seed images. K values greater than 0 include the original seed images with the k mutated seed images. Table 6 shows the coverage results of our coverage metrics on the mutated datasets with different k values:

Table 6: Coverage Results on Mutated Datasets with Different K Values

Model	K (Mutation strategy)	NBC	KMNC (K=3)	SNAC
ResNet-34	0	0.21486	0.36449	0.32940
	1	0.35385	0.37832	0.52878
	5	0.38556	0.40384	0.57601
VGG-16	0	0.17074	0.35715	0.24996
	1	0.26716	0.36684	0.38984
	5	0.32639	0.39757	0.47723

Table 6 presents the results of our coverage metric implementations on ResNet-34 with different k values for the mutation strategy. We used a k value of 3 for KMNC as determined in our earlier experiments. As shown in Table 6, the

use of mutation strategy with k=1, 5 improved the coverage results for all metrics as compared to the base pre-trained model. Specifically, for ResNet-34, the NBC coverage improved from 0.34382 to 0.35385 and 0.38556 respectively, while the KMNC (k = 3) coverage improved from 0.36449 to 0.37832 and 0.40384 respectively. The SNAC coverage also improved from 0.32940 to 0.52878 and 0.57601 respectively. Similar trends are seen for VGG-16, the NBC coverage increased from 0.23698 to 0.26716 and 0.32639, while KMNC (k = 3) increased from 0.35403 to 0.36684 and 0.39757. The SNAC coverage also improved from 0.34765 to 0.38984 and 0.47723.

Table 7: Coverage results percent difference – baseline vs mutated seeds

Model	K	NBC (%)	KMNC (%)	SNAC (%)
ResNet-34	0	-34.51	-0.24	-35.74
	1	2.92	3.55	3.16
	5	12.14	10.54	12.37
VGG-16	0	-27.95	0.88	-28.10
	1	12.74	3.62	12.14
	5	37.73	12.30	37.27

Table 7 presents a comparative analysis of the coverage results between the baseline and mutated seeds across the ResNet-34 and VGG-16 models. The table provides the percent differences in Neuron Boundary Coverage (NBC), K-multisection Neuron Coverage (KMNC), and Strong Neuron Activation Coverage (SNAC) metrics for various k values, illustrating the influence of the mutation strategy on the coverage results.

When the original seed data was fully replaced with mutation data (k = 0) for the ResNet-34 model, the coverage results experienced a decline for NBC (-34.51%) and SNAC (-35.74%), while a slight decrease was observed for KMNC (-0.24%). However, as the k value increased to 1, the coverage results improved, yielding positive percent differences in all three metrics: NBC (2.92%), KMNC (3.55%), and SNAC (3.16%). Further amplification in the k value to 5 led to even greater improvements in coverage results: NBC (12.14%), KMNC (10.54%), and SNAC (12.37%).

A similar trend emerged for the VGG-16 model. At k = 0, the coverage results revealed a decrease in NBC (-27.95%) and SNAC (-28.10%), whereas a slight increase was registered for KMNC (0.88%). When the k value was set to 1, the coverage results demonstrated a considerable improvement for all metrics: NBC (12.74%), KMNC (3.62%), and SNAC (12.14%). Subsequently, increasing the k value to 5 resulted in even more pronounced enhancements in coverage results: NBC (37.73%), KMNC (12.30%), and SNAC (37.27%).

Table 8: Model accuracy – baseline vs mutated seeds

Model	Accuracy (baseline)	Accuracy (mutated k = 5)	Percent Difference
ResNet - 34	89.479%	92.835%	3.75%
VGG - 16	86.725%	90.312%	4.14%

Table 8 showcases the impact of the mutation strategy (k = 5) on the performance of ResNet-34 and VGG-16, by comparing their accuracies before and after the mutation of seed data is used to fine-tune the two models. The percent differences in accuracy elucidate the efficacy of the mutation strategy and its potential for enhancing model performance.

In the case of the ResNet-34 model, the initial baseline accuracy stood at 89.479%. When the mutation strategy was applied with k = 5, the accuracy witnessed an increase to 92.835%. This translates to a positive percent difference of 3.75%,

which underlines the mutation strategy's ability to bolster the model's performance in the context of the ResNet-34 architecture.

For the VGG-16 model, a similar trend was observed. The baseline accuracy of this model was 86.725%, which rose to 90.312% after implementing the mutation strategy with k = 5. This improvement corresponds to a 4.14% percent difference, lending further credence to the effectiveness of the mutation strategy for boosting the model's performance.

6 INTERPRETATION AND ANALYSIS

In this study, we aimed to evaluate the quality and reliability of deep learning models in self-driving cars through the implementation of a unique seed mutation strategy guided by structural coverage metrics and the structural similarity index measure (SSIM) on the ResNet-34 and VGG-16 models.

6.1 Coverage Metric Performance

The baseline results (Table 4) indicated that the NBC and SNAC metrics achieved relatively high coverage scores, with NBC being the most effective in capturing the boundary conditions, and SNAC excelling at identifying strong neuron activations. In contrast, the KMNC metric had a lower coverage score, implying a potentially limited ability to capture the robustness of the models. This may be attributed to its sensitivity to the chosen k value, which defines the number of sections into which each neuron's activation range is divided.

Our experiments with varying k values for the KMNC metric (Table 5) revealed that lower k values resulted in higher coverage scores, suggesting that the KMNC metric is sensitive to the choice of k value. This highlights the importance of selecting an appropriate value to maximize coverage and maintain a balance between granularity and computational complexity.

6.2 Mutation Strategy Improvement

The implementation of our mutation strategy, which incorporated a uniform distribution for seed selection and SSIM for semantic similarity evaluation, demonstrated significant improvements in coverage results for all metrics when compared to the base pre-trained model.

A thorough examination of the results presented in Table 6 yields several noteworthy insights. For all coverage metrics, an increase in coverage results is observed as the k value, which represents the number of mutated seed images, rises. This observation implies that the mutated seed images effectively enhance the activation proportion of the deep learning models, culminating in more reliable and predictable model behaviour.

Additionally, an intriguing insight to consider is the decrease in coverage metric results for the models when the original seed images are substituted with their mutated equivalents (k = 0). A potential explanation for this behaviour may be attributed to the limited size of the original dataset and the fact that the models had been fine-tuned based on the original seed images. Consequently, the fine-tuned model weights may have failed to encapsulate substantial generalized feature information from the original seed images. This deficiency in generalization could be the underlying cause for the decline in coverage metric results when the models are introduced to the k = 0 mutated seed images. To address this issue, it may be advisable to either utilize a larger dataset for the initial fine-tuning of the pre-trained models or to generate mutated seed images and incorporate them into the original dataset during the initial fine-tuning phase. This approach would facilitate greater feature generalization to be captured by the pre-trained models, potentially improving the model performance on mutated datasets.

A key advantage of utilizing structural coverage metrics lies in their ability to quantitatively assess the extent to which a deep learning model's neurons and layers are activated within its intended environment. This evaluation offers a more comprehensive understanding of the model's performance, considering not only accuracy rates but also the activation landscape of the model's internal components.

For instance, a deep learning model with a high production accuracy rate may only have a small percentage of its neurons or layers activated during operation. This scenario leaves a significant portion of the model dormant, potentially leading to unexpected behaviours when subjected to adversarial inputs or out-of-distribution samples. As such, even with high production accuracy, a model with low activation levels may not be considered reliable or of high quality due to the potential unknown risks associated with the inactive portion of the model.

By employing these structural coverage metrics, we were able to accurately measure the activation levels of deep learning models. This knowledge can aid in identifying underutilized components and subsequently devising unique and innovative strategies to increase model activation in future works.

7 FUTURE CONSIDERATIONS AND CONCLUSION

While our study offers valuable insights into the effectiveness of coverage metrics for evaluating the reliability of deep learning models in self-driving cars, there are some limitations that warrant further investigation in future research.

Firstly, our experiments focused solely on the ResNet-34 and VGG-16 models. To substantiate our findings, it is essential to examine a broader range of deep learning architectures, encompassing newer models with advanced attention mechanisms such as transformers or capsule networks. Addressing hardware limitations is also crucial when conducting experimental testing in deep learning. Leveraging powerful GPU resources can significantly reduce processing time during testing, especially when handling larger image sizes. Future work could attempt to utilize the full 400x400 image sizes of the seed data, enabling more detailed feature extraction and richer generalization in the learning process.

Secondly, assessing the impact of various mutation strategies on coverage results is key to determining the most effective approach for enhancing the adversarial robustness of deep learning models. Investigating alternative mutation operators, as well as the combination of multiple operators, may shed light on the optimal strategies for improving model resilience. Moreover, exploring different similarity metrics for image comparison is a crucial future consideration. Although we employed SSIM in this study due to its alignment with our objectives, numerous other similarity metrics, such as vector norms, Feature-based similarity (FSIM), and Structural Dissimilarity Index (DSSIM), could potentially yield better results and warrant further exploration.

Additionally, future research should investigate the influence of dataset diversity and size on the effectiveness of coverage metrics and examine the applicability of our findings to other domains beyond autonomous vehicles. Incorporating real-world data from actual autonomous vehicle deployments would offer valuable insights into the practical robustness of deep learning systems under a wide array of challenging conditions, ensuring a comprehensive understanding of model performance in real-world scenarios.

REFERENCES

[1] 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. DeepGuage: Multi-Granularity Testing Criteria for Deep Learning Systems. ASE '18, (September 3-7, 2018), pp. 1

[2] 2023. "Self-Driving Cars with Convolutional Neural Networks (CNN)". https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn.

[3] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180220

[4] Ian j. GoodFellow, Jonathon Shlens and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples, ICLR 2015, (March 20, 2015), pp. 1

[5] 2014. "This Is How Bad Self-Driving Cars Suck In The Rain". http://jalopnik.com/ this-is-how-bad-self-driving-cars-suck-in-the-rain-1666268433.

[6] 2022. "Tesla behind eight-vehicle crash was in 'full self-driving' mode, says driver". https://www.theguardian.com/technology/2022/dec/22/tesla-crash-full-self-driving-mode-san-francisco.

[7] Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: Automated whitebox testing of deep learning systems. In: SOSP (2017)

[8] Z. Soleimanitaleb, M. Ali Keyvanrad, A. Jafari. "Object Tracking Methods: A Review". in *9th International Conference on Computer and Knowledge Engineering (ICCKE 2019).* October 24-25 2019. pp. 282

[9] N. Barla. V7labs March 19 2022. "The Complete Guide to Object Tracking [+V7 Tutorial]. https://www.v7labs.com/blog/object-tracking-guide

[10] H. Xu, Y, Zhu. "Real-time object tracking based on improved fully-convolutional siamese network". in *Computers & Electrical Engineering (Elsevier)*. September 2020. pp.1

[11] Z. Soleimanitaleb, M. Ali Keyvanrad, A. Jafari. "Object Tracking Methods: A Review". in *9th International Conference on Computer and Knowledge Engineering (ICCKE 2019).* October 24-25 2019. pp. 283

[12] M. Isaksson. Towards Data Science June 6, 2020. "Four Common Types of Neural Network Layers". https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c

[13] Robust discrimination of human footsteps using seismic signals - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-structure-of-a-multilayer-perceptron-neural-network_fig3_241347660 [accessed 23 Apr, 2023]

[14] Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: Automated whitebox testing of deep learning systems. In: SOSP (2017)

[15] 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. DeepGuage: Multi-Granularity Testing Criteria for Deep Learning Systems. ASE '18, (September 3-7, 2018)

[16] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3293882.3330579

[17] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing. (March 7, 2018)

[18] 3D Object Representations for Fine-Grained Categorization. Jonathan Krause, Michael Stark, Jia Deng, Li Fei-Fei. 4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013 (3dRR-13). Sydney, Australia. Dec. 8, 2013.

[19] Cars Dataset. https://ai.stanford.edu/~jkrause/cars/car_dataset.html

[20] 2023. "Deep Residual Networks (ResNet, ResNet50) – 2023 Guide". https://viso.ai/deep-learning/resnet-residual-neural-network/

[21] 2021. "Everything you need to know about VGG16". https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. (Dec 10, 2015)

[23] Muhammad Usman, Youcheng Sun, Divya Gopinath, Rishi Dange, Luca Manolache and Corina S. Pa˘s˘areanu. 2022. An Overview of Structural Coverage Metrics for Testing Neural Networks. (August 5, 2022)

[24] Zhou Wang, Member, IEEE, Alan Conrad Bovik, Fellow, IEEE, Hamid Rahim Sheikh, Student Member, IEEE, and Eero P. Simoncelli, Senior Member, IEEE. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. (April 4, 2004)