Title

Counterexample-Guided Data Augmentation

ISBN
978-0-9992411-2-7

Authors
Dreossi, Tommaso
Ghosh, Shromona
Yue, Xiangyu
et al.

Publication Date
2018

Peer reviewed

Counterexample-Guided Data Augmentation

Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue,
Kurt Keutzer, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia
University of California, Berkeley
{dreossi | shromona.ghosh | xyyue}@berkeley.edu,
{keutzer | alberto | sseshia}@eecs.berkeley.edu

Abstract

We present a novel framework for augmenting data sets for machine learning based on *counterexamples*. Counterexamples are misclassified examples that have important properties for retraining and improving the model. Key components of our framework include a *counterexample generator*, which produces data items that are misclassified by the model and *error tables*, a novel data structure that stores information pertaining to misclassifications. Error tables can be used to explain the model's vulnerabilities and are used to efficiently generate counterexamples for augmentation. We show the efficacy of the proposed framework by comparing it to classical augmentation techniques on a case study of object detection in autonomous driving based on deep neural networks.

1 Introduction

Models produced by machine learning algorithms, especially *deep neural networks*, are being deployed in domains where trustworthiness is a big concern, creating the need for higher accuracy and assurance [Russell *et al.*, 2015; Seshia *et al.*, 2016]. However, learning high-accuracy models using deep learning is limited by the need for large amounts of data, and, even further, by the need of labor-intensive labeling. *Data augmentation* overcomes the lack of data by inflating training sets with label-preserving transformations, i.e., transformations which do not alter the label. Traditional data augmentation schemes [van Dyk and Meng, 2001; Simard *et al.*, 2003; Cireşan *et al.*, 2011; Ciregan *et al.*, 2012; Krizhevsky *et al.*, 2012] involve geometric transformations which alter the geometry of the image (e.g., rotation, scaling, cropping or flipping); and photometric transformations which vary color channels. The efficacy of these techniques have been demonstrated recently (see, e.g., [Xu *et al.*, 2016; Wong *et al.*, 2016]). Traditional augmentation schemes, like the aforementioned methods, add data to the training set hoping to improve the model accuracy without taking into account what kind of features the model has already learned. More recently, a sophisticated data augmentation technique has been proposed [Liang *et al.*, 2017; Marchesi, 2017] which uses Generative Adversarial Networks [Goodfellow *et al.*, 2014], a particular kind of neural network able to generate synthetic data, to inflate training sets. There are also augmentation techniques, such as hard negative mining [Shrivastava *et al.*, 2016], that inflate the training set with targeted negative examples with the aim of reducing false positives.

In this work, we propose a new augmentation scheme, *counterexample-guided data augmentation*. The main idea is to augment the training set only with new misclassified examples rather than modified images coming from the original training set. The proposed augmentation scheme consists of the following steps: 1) Generate synthetic images that are misclassified by the model, i.e., the counterexamples; 2) Add the counterexamples to the training set; 3) Train the model on the augmented dataset. These steps can be repeated until the desired accuracy is reached. Note that our augmentation scheme depends on the ability to generate misclassified images. For this reason, we developed an *image generator* that cooperates with a *sampler* to produce images that are given as input to the model. The images are generated in a manner such that the ground truth labels can be automatically added. The incorrectly classified images constitute the augmentation set that is added to the training set. In addition to the pictures, the image generator provides information on the misclassified images, such as the disposition of the elements, brightness, contrast, etc. This information can be used to find features that frequently recur in counterexamples. We collect information about the counterexamples in a data structure we term as the "*error table*". Error tables are extremely useful to provide *explanations* about counterexamples and find recurring patterns that can lead an image to be misclassified. The error table analysis can also be used to generate images which are likely to be counterexamples, and thus, efficiently build augmentation sets.

In summary, the main contributions of this work are:

- A *counterexample-guided data augmentation* approach where only misclassified examples are iteratively added to training sets;
- A synthetic *image generator* that renders realistic counterexamples;
- *Error tables* that store information about counterexamples and whose analysis provides explanations and facilitates the generation of counterexample images.

We conducted experiments on Convolutional Neural Net-

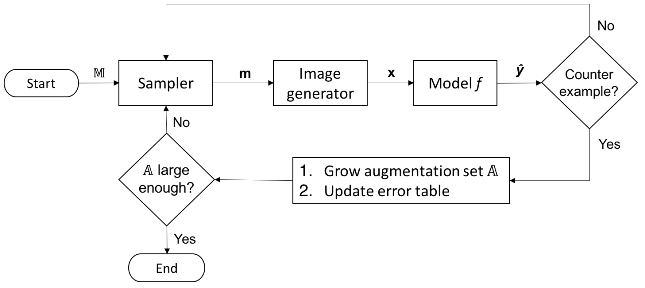


Figure 1: Counterexample-guided augmentation scheme. $\mathbb{M}$ denotes the modification space, $\mathbf{m}$ is a sampled modification, $\mathbf{x}$ is the image generated from $\mathbf{m}$, $\hat{\mathbf{y}}$ is the model prediction.

works (CNNs) for object detection by analyzing different counterexample data augmentation sampling schemes and compared the proposed methods with classic data augmentation. Our experiments show the benefits of using a counterexample-driven approach against a traditional one. The improvement comes from the fact that a counterexample augmentation set contains information that the model had not been able to learn from the training set, a fact that was not considered by classic augmentation schemes. In our experiments, we use synthetic data sets generated by our image generator. This ensures that all treated data comes from the same distribution.

Overview

Fig. 1 summarizes the proposed counterexample-guided augmentation scheme. The procedure takes as input a modification space, $\mathbb{M}$, the space of possible configurations of our image generator. The space $\mathbb{M}$ is constructed based on domain knowledge to be a space of "semantic modifications;" i.e., each modification must have a meaning in the application domain in which the machine learning model is being used. This allows us to perform more meaningful data augmentation than simply through adversarial data generation performed by perturbing an input vector (e.g., adversarially selecting and modifying a small number of pixel values in an image).

In each loop, the sampler selects a modification, $\mathbf{m}$, from $\mathbb{M}$. The sample is determined by a sampling method that can be biased by a precomputed *error table*, a data structure that stores information about images that are misclassified by the model. The sampled modification is rendered into a picture $\mathbf{x}$ by the image generator. The image $\mathbf{x}$ is given as input to the model f that returns the prediction $\hat{\mathbf{y}}$. We then check whether $\mathbf{x}$ is a counterexample, i.e., the prediction $\hat{\mathbf{y}}$ is wrong. If so, we add $\mathbf{x}$ to our augmentation set $\mathbb{A}$ and we store $\mathbf{x}$'s information (such as $\mathbf{m}$, $\hat{\mathbf{y}}$) in the error table that will be used by the sampler at the next iteration. The loop is repeated until the augmentation set $\mathbb{A}$ is large enough (or $\mathbb{M}$ has been sufficiently covered).

This scheme returns an *augmentation set*, that will be used to retrain the treated model, along with an *error table*, whose analysis identifies common features among counterexamples and aids the sampler to select candidate counterexamples.

The paper structure mostly follows the scheme of Fig. 1: Sec. 2 introduces some notation; Sec. 3 describes the image generator used to render synthetic images; Sec. 4 introduces some sampling techniques that can be used to efficiently sample the modification space; Sec. 5 introduces error tables and details how they can be used to provide explanations about counterexamples; Sec. 6 concludes the paper by evaluating the proposed techniques and comparing across different tunings of our counterexample-guided augmentation scheme and the proposed methods against classic augmentation. The implementation of the proposed framework and the reported experiments are available at `https://github.com/dreossi/analyzeNN`.

2 Preliminaries

This section provides the notation used throughout this paper.

Let $\mathbf{a}$ be a vector, a_i be its i-th element with index starting at $i = 1$, $a_{i:j}$ be the range of elements of $\mathbf{a}$ from i to j; and $\mathbb{A}$ be a set. $\mathbb{X}$ is a set of training examples, $\mathbf{x}^{(i)}$ is the i-th example from a dataset and $\mathbf{y}^{(i)}$ is the associated label. $f : \mathbb{A} \to \mathbb{B}$ is a model (or function) f with domain $\mathbb{A}$ and range $\mathbb{B}$. $\hat{\mathbf{y}} = f(\mathbf{x})$ is the prediction of the model f for input $\mathbf{x}$. In the object detection context, $\hat{\mathbf{y}}$ encodes bounding boxes, scores, and categories predicted by f for the image $\mathbf{x}$. $f_{\mathbb{X}}$ is the model f trained on $\mathbb{X}$. Let B_1 and B_2 be bounding boxes encoded by $\hat{\mathbf{y}}$. The *Intersection over Union* (IoU) is defined as $IoU(A_{B_1}, A_{B_2}) = A_{B_1} \cap A_{B_2} / A_{B_1} \cup A_{B_2}$, where A_{B_i} is the area of B_i, with $i \in \{1, 2\}$. We consider $B_{\hat{\mathbf{y}}}$ to be a *detection* for $B_{\mathbf{y}}$ if $IoU(B_{\hat{\mathbf{y}}}, B_{\mathbf{y}}) > 0.5$. *True positives* tp is the number of correct detections; *false positives* fp is the number of predicted boxes that do not match any ground truth box; *false negatives* is the number of ground truth boxes that are not detected.

Precision and *recall* are defined as $p(\hat{\mathbf{y}}, \mathbf{y}) = tp/(tp + fp)$ and $r(\hat{\mathbf{y}}, \mathbf{y}) = tp/(tp + fn)$. In this work, we consider an input $\mathbf{x}$ to be *misclassified* if $p(\hat{\mathbf{y}}, \mathbf{y})$ or $r(\hat{\mathbf{y}}, \mathbf{y})$ is less than 0.75. Let $\mathbb{T} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$ be a test set with m examples. The *average precision* and *recall* of f are defined as $ap_f(\mathbb{T}) = \frac{1}{m}\sum_{i=1}^{m} p(f(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$ $ar_f(\mathbb{T}) = \frac{1}{m}\sum_{i=1}^{m} r(f(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$. We use average precision and recall to measure the accuracy of a model, succinctly represented as $acc_f(\mathbb{T}) = (ap_f(\mathbb{T}), ar_f(\mathbb{T}))$.

3 Image Generator

At the core of our counterexample augmentation scheme is an image generator (similar to the one defined in [Dreossi *et al.*, 2017a; 2017b]) that renders realistic synthetic images of road scenarios. Since counterexamples are generated by the synthetic data generator, we have full knowledge of the ground truth labels for the generated data. In our case, for instance, when the image generator places a car in a specific position, we know exactly its location and size, hence the ground truth bounding box is accordingly determined. In this section, we describe the details of our image generator.

3.1 Modification Space

The image generator implements a *generation function* $\gamma : \mathbb{M} \to \mathbb{X}$ that maps every modification $\mathbf{m} \in \mathbb{M}$ to a feature

$\gamma(\mathbf{m}) \in \mathbb{X}$. Intuitively, a modification describes the configuration of an image. For instance, a three-dimensional modification space can characterize a car x (lateral) and z (away) displacement on the road and the image brightness. A generator can be used to abstract and compactly represent a subset of a high-dimensional image space.

We implemented an image generator based on a 14D modification space whose dimensions determine a road background; number of cars (one, two or three) and their x and z positions on the road; brightness, sharpness, contrast, and color of the picture. Fig. 2 depicts some images rendered by our image generator.

We can define a metric over the modification space to measure the diversity of different pictures. Intuitively, the distance between two configurations is large if the concretized images are visually diverse and, conversely, it is small if the concretized images are similar.

The following is an example of metric distance that can be defined over our 14D modification space. Let $\mathbf{m}^{(1)}, \mathbf{m}^{(2)} \in \mathbb{M}$ be modifications. The distance is defined as:

$$d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = \sum_{i=1}^{4} \mathbf{1}_{m_i^{(1)} \neq m_i^{(2)}} + \|m^{(1)}_{5:14} - m^{(2)}_{5:14}\| \tag{1}$$

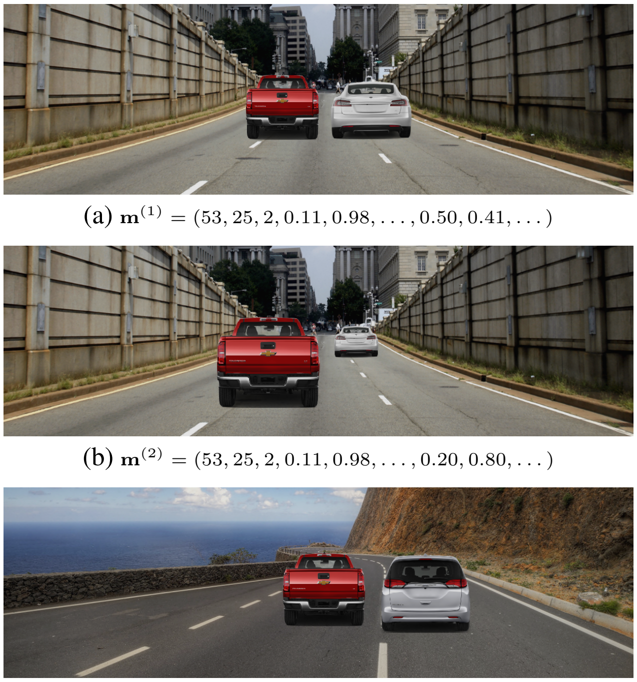
where $\mathbf{1}_{condition}$ is 1 if the condition is true, 0 otherwise, and $\|\cdot\|$ is the L^2 norm. The distance counts the differences between background and car models and adds the Euclidean distance of the points corresponding to x and z positions, brightness, sharpness, contrast, and color of the images.

Fig. 2 depicts three images with their modifications $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$, and $\mathbf{m}^{(3)}$. For brevity, captions report only the dimensions that differ among the images, that are background, car models and x, z positions. The distances between the modifications are $d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$, $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$, $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$. Note how similar images, like Fig. 2 (a) and (b) (same backgrounds and car models, slightly different car positions), have smaller distance ($d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$) than diverse images, like Fig. (a) and (c); or (b) and (c) (different backgrounds, car models, and vehicle positions), whose distances are $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$ and $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$.

Later on, we use this metric to generate sets whose elements ensure a certain amount of diversity. (see Sec. 6.1)

3.2 Picture Concretization

Once a modification is fixed, our picture generator renders the corresponding image. The concretization is done by superimposing basic images (such as road background and vehicles) and adjusting image parameters (such as brightness, color, or contrast) accordingly to the values specified by the modification. Our image generator comes with a database of backgrounds and car models used as basic images. Our database consists of 35 road scenarios (e.g., desert, forest, or freeway scenes) and 36 car models (e.g., economy, family, or sports vehicles, from both front and rear views). The database can be easily extended or replaced by the user.



(a) $\mathbf{m}^{(1)} = (53, 25, 2, 0.11, 0.98, \ldots, 0.50, 0.41, \ldots)$

(b) $\mathbf{m}^{(2)} = (53, 25, 2, 0.11, 0.98, \ldots, 0.20, 0.80, \ldots)$

(c) $\mathbf{m}^{(3)} = (13, 25, 7, 0.11, 0.98, \ldots, 0.50, 0.41, \ldots)$

Figure 2: Distance over modification space used to measure visual diversity of concretized images. $d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$, $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$, $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$.

3.3 Annotation Tool

In order to render realistic images, the picture generator must place cars on the road and scale them accordingly. To facilitate the conversion of a modification point describing x and z position into a proper superimposition of the car image on a road, we equipped the image generator with an annotation tool that can be used to specify the sampling area on a road and the scaling factor of a vehicle. For a particular road, the user draws a trapezoid designating the area where the image generator is allowed to place a car. The user also specifies the scale of the car image on the trapezoid bases, i.e., at the closest and furthest points from the observer (see Fig. 3). When sampling a point at an intermediate position, i.e., inside the trapezoid, the tool interpolates the provided car scales and determines the scaling at the given point. Moreover, the image generator superimposes different vehicles respecting the perspective of the image. The image generator also performs several checks to ensure that the rendered cars are visible.

4 Sampling Methods

The goal of the sampler is to provide a good coverage of the modification space and identify samples whose concretizations lead to counterexamples.

We now briefly describe some sampling methods (similar to those defined in [Dreossi *et al.*, 2017a; 2017b]) that we integrated into our framework:

- *Uniform Random Sampling*: Uniform random sampling ensures an equal probability of sampling any possible

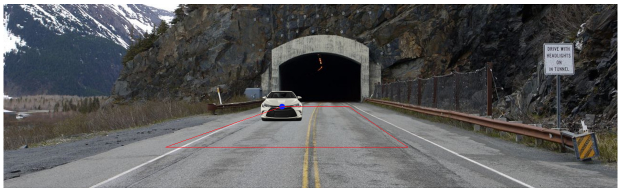


Figure 3: Annotation trapezoid. User adjusts the four corners that represent the valid sampling subspace of x and z. The size of the car scales according to how close it is to the vanishing point.

point from $\mathbb{M}$, which guarantees a good mix of generated images for both training and testing. Although a simple and effective technique for both training as well as testing, it may not provide a good coverage of the modification space;

- *Low-Discrepancy Sampling*: A *low-discrepancy* (or quasi-random) sequence is a sequence of *n*-tuples that fills a *n*D space more uniformly than uncorrelated random points. Low-discrepancy sequences are useful to cover boxes by reducing gaps and clustering of points which ensures uniform coverage of the sample space.
- *Cross-Entropy Sampling*: The *cross-entropy* method was developed as a general Monte Carlo approach to combinatorial optimization and importance sampling. It is a iterative sampling technique, where we sample from a a given probability distribution, and update the distribution by minimizing the cross-entropy.

Some examples of low-discrepancy sequences are the Van der Corput, Halton [Halton, 1960], or Sobol [Sobol, 1976] sequences. In our experiments, we use the Halton [Niederreiter, 1988] sequence. There are two main advantages in having optimal coverage: first, we increase the chances of quickly discovering counterexamples, and second, the set of counterexamples will have high diversity; implying the concretized images will look different and thus the model will learn diverse new features.

5 Error Tables

Every iteration of our augmentation scheme produces a counterexample that contains information pointing to a limitation of the learned model. It would be desirable to extract patterns that relate counterexamples, and use this information to efficiently generate new counterexamples. For this reason, we define *error tables* that are data structures whose columns are formed by important features across the generated images. The error table analysis is useful for:

1. Providing *explanations* about counterexamples, and
2. Generating *feedback* to sample new counterexamples.

In the first case, by finding common patterns across counterexamples, we provide feedback to the user like "*The model does not detect white cars driving away from us in forest roads*"; in the second case, we can bias the sampler towards modifications that are more likely to lead to counterexamples.

Table 1: Example of error table proving information about counterexamples. First rows describes Fig. 3. Implicit unordered features: car model, environment; explicit ordered features: brightness, x, z car coordinates; explicit unordered feature: background ID.

Car model	Background ID	Environment	Brightness	x	z
Toyota	12	Tunnel	0.9	0.2	0.9
BMW	27	Forest	1.1	0.4	0.7
Toyota	11	Forest	1.2	0.4	0.8

5.1 Error Table Features

We first provide the details of the kinds of features supported by our error tables. We categorize features along two dimensions:

1. *Explicit* vs. *implicit* features: Explicit features are sampled from the modification space (e.g., x, z position, brightness, contrast, etc.) whereas implicit features are user-provided aspects of the generated image (e.g., car model, background scene, etc.).
2. *Ordered* vs. *unordered* features: some features have a domain with a well-defined total ordering (e.g., sharpness) whereas others do not have a notion of ordering (e.g., car model, identifier of background scene, etc.).

The set of implicit and explicit features are mutually exclusive. In general, implicit features are more descriptive and characterize the generated images. These are useful for providing feedback to explain the vulnerabilities of the classifier. While implicit features are unordered, explicit features can be ordered or unordered. Rows of error tables are the realizations of the features for misclassification.

Tab. 1 is an illustrative error table. The table includes car model and environment scene (implicit unordered features), brightness, x, z car coordinates (explicit ordered features), and background ID (explicit unordered feature). The first row of Tab. 1 actually refers to Fig. 3. The actual error tables generated by our framework are larger than Tab. 1. They include, for instance, our 14D modification space (see Sec. 3.1) and features like number of vehicles, vehicle orientations, dominant color in the background, etc.

Given an error table populated with counterexamples, we would like to analyze it to provide feedback and utilize this feedback to sample new images.

5.2 Feature Analysis

A naive analysis technique is to treat all the features equally, and search for the most commonly occurring element in each column of the error table. However, this assumes no correlation between the features, which is often not the case. Instead, we develop separate analysis techniques for ordered and unordered features. In the following we discuss how we can best capture correlations between the two sets:

- *Ordered features*: Since these features are ordered, a meaningful analysis technique would be to find the direction in the feature space where most of the falsifying samples occur. This is very similar to model order reduction using Principal Component Analysis (PCA [Wold *et al.*, 1987]). Specifically, we are interested in the first principal component, which is the singular vector corresponding to the largest singular value in the Singular

Value Decomposition (SVD [Wold *et al.*, 1987]) of the matrix consisting of all the samples of the ordered features. We can use the singular vector to find how sensitive the model is with respect to the ordered features. If the value corresponding to a feature is small in the vector, it implies that the model is not robust to changes in that feature, i.e., changes in that feature would affect the misclassification. Or alternatively, features corresponding to larger values in the singular vector, act as "don't cares", i.e., by fixing all other features, the model misclassifies the image regardless the value of this feature;

- *Unordered features*: Since these features are unordered, their value holds little importance. The most meaningful information we can gather from them is the subsets of features which occurs together most often. To correctly capture this, we must explore all possible subsets, which is a combinatorial problem. This proves to be problematic when the space of unordered features is large. One way to overcome this is by limiting the size of the maximum subset to explore.

We conducted an experiment on a set of 500 counterexamples. The ordered features included x and z positions of each car; along with the brightness, contrast, sharpness, and color of the overall image. The explicit features include the ordered features along with the discrete set of all possible cars and backgrounds. The implicit features include details like color of the cars, color of the background, orientation of the cars, etc. The PCA on the explicit ordered features revealed high values corresponding to the x position of the first car (0.74), brightness (0.45) and contrast (0.44). We can conclude that the model is not robust to changes in these ordered features. For the unordered features, the combination of forest road with one white car with its rear towards the camera and the other cars facing the camera, appeared 13 times. This provides an explanation of recurrent elements in counterexamples, specifically "*The model does not detect white cars driving away from us in forest roads*".

5.3 Sampling Using Feedback

We can utilize the feedback provided by the error table analysis to guide the sampling for subsequent training. Note that we can only sample from the explicit features:

- *Feedback from Ordered Features*: The ordered features, which is a subset of the explicit features, already tell us which features need to vary more during the sampling process. For example, in the example of Sec. 5.2, our sampler must prioritize sampling different x positions for the first car, then brightness, and finally contrast among the other ordered features;
- *Feedback from Unordered Features*: Let $\mathbb{S}_{uf} = \mathbb{S}_{ef} \cup \mathbb{S}_{if}$ be the subset of most occurring unordered features returned by the analysis, where $\mathbb{S}_{ef}$ and $\mathbb{S}_{if}$ are the mutually exclusive sets of explicit and implicit features, respectively. The information of $\mathbb{S}_{ef}$ can be directly incorporated into the sampler. The information provided by $\mathbb{S}_{if}$ require some reasoning since implicit features are not directly sampled. However, they are associated with particular elements of the image (e.g., background or vehicle).

Table 2: Comparison of augmentation techniques. Precisions (top) and recalls (bottom) are reported. $\mathbb{T}_T$ set generated with sampling method T; $f_{\mathbb{X}_T}$ model f trained on $\mathbb{X}$ augmented with technique $T \in \{S, R, H, C, D, M\}$; S: standard, R: uniform random, H: low-discrepancy Halton, C: cross-entropy, D: uniform random with distance constraint, M: mix of all methods.

	$\mathbb{T}_R$	$\mathbb{T}_H$	$\mathbb{T}_C$	$\mathbb{T}_D$	$\mathbb{T}_M$
$f_{\mathbb{X}}$	0.6169 0.7429	0.6279 0.7556	0.3723 0.4871	0.7430 0.8373	0.6409 0.7632
$f_{\mathbb{X}_S}$	0.6912 0.8080	0.6817 0.7987	0.3917 0.5116	0.7824 0.8768	0.6994 0.8138
$f_{\mathbb{X}_R}$	0.7634 0.8667	0.7515 0.8673	0.5890 0.7242	0.8484 0.9745	0.7704 0.8818
$f_{\mathbb{X}_H}$	0.7918 0.8673	0.7842 0.8727	0.5640 0.6693	0.8654 0.9598	0.7980 0.8828
$f_{\mathbb{X}_C}$	0.7778 0.7804	0.7632 0.7722	0.6140 0.7013	0.8673 0.8540	0.7843 0.7874
$f_{\mathbb{X}_D}$	0.7516 0.8642	0.7563 0.8724	0.6057 0.7198	0.8678 0.9612	0.7670 0.8815

We can use the image generator library and error table to recognize which elements in the library the components of $\mathbb{S}_{if}$ correspond to, and set the feature values accordingly. For instance, in the example of Sec. 5.2, the analysis of the unordered explicit features revealed that the combination of bridge road with a Tesla, Mercedes, and Mazda was most often misclassified. We used this information to generate more images with this particular combination by varying brightness and contrast.

Sec 6.3 shows how this technique leads to a larger fraction of counterexamples that can be used for retraining.

6 Experimental Evaluation

In this section, we show how the proposed techniques can be used to augment training sets and improve the accuracy of the considered models. We will experiment with different sampling methods, compare counterexample guided augmentation against classic augmentation, iterate over several augmentation cycles, and finally show how error tables are useful tools for analyzing models. The implementation of the proposed framework and the reported experiments are available at `https://github.com/dreossi/analyzeNN`.

In all the experiments we analyzed squeezeDet [Wu *et al.*, 2016], a CNN real-time object detector for autonomous driving. All models were trained for 65 epochs.

The original training and test sets $\mathbb{X}$ and $\mathbb{T}$ contain 1500 and 750 pictures, respectively, randomly generated by our image generator. The initial accuracy $acc_{f_{\mathbb{X}}}(\mathbb{T}) = (0.9847, 0.9843)$ is relatively high (see Tab. 3). However, we will be able to generate sets of counterexamples as large as $\mathbb{T}$ on which the accuracy of $f_{\mathbb{X}}$ drops down. The highlighted entries in the tables show the best performances. Reported values are the averages across five different experiments.

6.1 Augmentation Methods Comparison

As the first experiment, we run the counterexample augmentation scheme using different sampling techniques (see Sec. 4). Specifically, we consider uniform random sampling, low-discrepancy Halton sequence, cross-entropy sampling, and uniform random sampling with a diversity constraint on the sampled points. For the latter, we adopt the distance defined in Sec. 3.1 and we require that the modifications of the counterexamples must be at least distant by 0.5 from each other.

For every sampling method, we generate 1500 counterexamples, half of which are injected into the original training set $\mathbb{X}$ and half are used as test sets. Let R, H, C, D denote uniform random, Halton, cross-entropy, and diversity (i.e., random with distance constraint) sampling methods. Let $S \in \{R, H, C, D\}$ be a sampling technique. $\mathbb{X}_S$ is the augmentation of $\mathbb{X}$, and $\mathbb{T}_S$ is a test set, both generated using S. For completeness, we also defined the test set $\mathbb{T}_M$ containing an equal mix of counterexamples generated by all the R, H, C, D sampling methods.

Tab. 2 reports the accuracy of the models trained with various augmentation sets evaluated on test sets of counterexamples generated with different sampling techniques. The first row reports the accuracy of the model $f_{\mathbb{X}}$ trained on the original training set $\mathbb{X}$. Note that, despite the high accuracy of the model on the original test set ($acc_{f_{\mathbb{X}}}(\mathbb{T}) = (0.9847, 0.9843)$), we were able to generate several test sets from the same distribution of $\mathbb{X}$ and $\mathbb{T}$ on which the model poorly performs.

The first augmentation that we consider is the standard one, i.e., we alter the images of $\mathbb{X}$ using `imgaug`[1], a Python library for images augmentation. We augmented 50% of the images in $\mathbb{X}$ by randomly cropping $10 - 20\%$ on each side, flipping horizontally with probability 60%, and applying Gaussian blur with $\sigma \in [0.0, 3.0]$. Standard augmentation improves the accuracies on every test set. The average precision and recall improvements on the various test sets are 4.91% and 4.46%, respectively (see Row 1 Tab. 2).

Next, we augment the original training set $\mathbb{X}$ with our counterexample-guided schemes (uniform random, low-discrepancy Halton, cross-entropy, and random with distance constraint) and test the retrained models on the various test sets. The average precision and recall improvements for uniform random are 14.43% and 14.56%, for low-discrepancy Halton 16.05% and 14.57%, for cross-entropy 16.11% and 6.18%, and for random with distance constraint 14.95% and 14.26%. First, notice the improvement in the accuracy of the original model using counterexample-guided augmentation methods is larger compared to the classic augmentation method. Second, among the proposed techniques, cross-entropy has the highest improvement in precision but low-discrepancy tends to perform better than the other methods in average for both precision and recall. This is due to the fact that low-discrepancy sequences ensure more diversity on the samples than the other techniques, resulting in different pictures from which the model can learn new features or reinforce the weak ones.

The generation of a counterexample for the original model $f_{\mathbb{X}}$ takes in average for uniform random sampling 30s, for Halton 92s, and for uniform random sampling with constraints 55s. This shows the trade-off between time and gain in model accuracy. The maximization of the diversity of the augmentation set (and the subsequent accuracy increase) requires more iterations.

6.2 Augmentation Loop

For this experiment, we consider only the uniform random sampling method and we incrementally augment the training set over several augmentation loops. The goal of this experiments is to understand whether the model overfits the counterexamples and see if it is possible to reach a saturation point, i.e., a model for which we are not able to generate counterexamples. We are also interested in investigating the relationship between the quantity of injected counterexamples and the accuracy of the model.

Consider the i-th augmentation cycle. For every augmentation round, we generate the set of counterexamples by considering the model $f_{\mathbb{X}^{[i]}_r}$ with highest average precision and recall. Given $\mathbb{X}^{[i]}$, our analysis tool generates a set $\mathbb{C}^{[i]}$ of counterexamples. We split $\mathbb{C}^{[i]}$ in halves $\mathbb{C}^{[i]}_{\mathbb{X}}$ and $\mathbb{C}^{[i]}_{\mathbb{T}}$. We use $\mathbb{C}^{[i]}_{\mathbb{X}}$ to augment the original training set $\mathbb{X}^{[i]}$ and $\mathbb{C}^{[i]}_{\mathbb{T}}$ as a test set. Specifically, the augmented training set $\mathbb{X}^{[i+1]}_{r'} = \mathbb{X}^{[i]}_r \cup \mathbb{C}^{[i]}_{\mathbb{X}}$ is obtained by adding misclassified images of $\mathbb{C}^{[i]}_{\mathbb{X}}$ to $\mathbb{X}^{[i]}$. r, r' are the ratios of misclassified images to original training examples. For instance, $|\mathbb{X}_{0.08}| = |\mathbb{X}| + 0.08 * |\mathbb{X}|$, where $|\mathbb{X}|$ is the cardinality of $\mathbb{X}$. We consider the ratios $0.08, 0.17, 0.35, 0.50$. We evaluate every model against every test set.

Tab. 3 shows the accuracies for three augmentation cycles. For each model, the table shows the average precision and recall with respect to the original test set $\mathbb{T}$ and the tests sets of misclassified images. The generation of the first loop took around 6 hours, the second 14 hours, the third 26 hours. We stopped the fourth cycle after more than 50 hours. This shows how *it is increasingly hard to generate counterexamples for models trained over several augmentations*. This growing computational hardness of counterexample generation with the number of cycles is an informal, empirical measure of increasing assurance in the machine learning model.

Notice that for every cycle, our augmentation improves the accuracy of the model with respect to the test set. Even more interesting is the fact that the model accuracy on the original test set does not decrease, but actually improves over time (at least for the chosen augmentation ratios).

6.3 Error Table-Guided Sampling

In this last experimental evaluation, we use error tables to analyze the counterexamples generated for $f_{\mathbb{X}}$ with uniform random sampling. We analyzed both the ordered and unordered features (see Sec. 5.2). The PCA analysis of ordered features revealed the following relevant values: sharpness 0.28, contrast 0.33, brightness 0.44, and x position 0.77. This tells us that the model is more sensitive to image alterations rather than to the disposition of its elements. The occurrence counting of unordered features revealed that the top three most occurring car models in misclassifications are white Porsche, yellow Corvette, and light green Fiat. It is interesting to note that all these models have uncommon designs if compared to popular cars. The top three most recurring background scenes are a narrow bridge in a forest, an indoor parking lot, and a downtown city environment. All these scenarios are characterized by a high density of details that lead to false

[1] imgaug: `https://github.com/aleju/imgaug`

Table 3: Augmentation loop. For the best (highlighted) model, a test set $\mathbb{C}_{\mathbb{T}}^{[i]}$ and augmented training set $\mathbb{X}_r^{[i+1]}$ are generated. r is the ratio of counterexamples to the original training set.

	$\mathbb{T}$	$\mathbb{C}_{\mathbb{T}}^{[1]}$	$\mathbb{C}_{\mathbb{T}}^{[2]}$	$\mathbb{C}_{\mathbb{T}}^{[3]}$
$f_{\mathbb{X}}$	0.9847 0.9843	0.6957 0.7982		
$f_{\mathbb{X}_{0.08}^{[1]}}$	0.9842 0.9861	0.7630 0.8714		
$f_{\mathbb{X}_{0.17}^{[1]}}$	0.9882 0.9905	0.8197 0.9218	0.5922 0.8405	
$f_{\mathbb{X}_{0.35}^{[1]}}$	0.9843 0.9906	0.8229 0.9110		
$f_{\mathbb{X}_{0.50}^{[1]}}$	0.9872 0.9912	0.7998 0.9149		
$f_{\mathbb{X}_{0.08}^{[2]}}$	0.9947 0.9955	0.7286 0.8691	0.7159 0.8612	
$f_{\mathbb{X}_{0.17}^{[2]}}$	0.9947 0.9954	0.7424 0.8422	0.7288 0.8628	
$f_{\mathbb{X}_{0.35}^{[2]}}$	0.9926 0.9958	0.7732 0.8720	0.7570 0.8762	
$f_{\mathbb{X}_{0.50}^{[2]}}$	0.9900 0.9942	0.8645 0.9339	0.8223 0.9187	0.5308 0.7017
$f_{\mathbb{X}_{0.08}^{[3]}}$	0.9889 0.9972	0.7105 0.8571	0.7727 0.8987	0.7580 0.8860
$f_{\mathbb{X}_{0.17}^{[3]}}$	0.9965 0.9970	0.8377 0.9116	0.8098 0.9036	0.8791 0.9473
$f_{\mathbb{X}_{0.35}^{[3]}}$	0.9829 0.9937	0.7274 0.8060	0.8092 0.8816	0.7701 0.8480
$f_{\mathbb{X}_{0.50}^{[3]}}$	0.9905 0.9955	0.7513 0.8813	0.7891 0.9573	0.7902 0.9169

positives. Using the gathered information, we narrowed the sampler space to the subsets of the modification space identified by the error table analysis. The counterexample generator was able to produce 329 misclassification with 10k iterations, against the 103 of pure uniform random sampling, 287 of Halton, and 96 of uniform random with distance constraint.

Finally, we retrained f on the training set $\mathbb{X}_E$ that includes 250 images generated using the error table analysis. The obtained accuracies are $acc_{f_{\mathbb{X}_E}}(\mathbb{T}_R) = (0.7490, 0.8664)$, $acc_{f_{\mathbb{X}_E}}(\mathbb{T}_H) = (0.7582, 0.8751)$, $acc_{f_{\mathbb{X}_E}}(\mathbb{T}_D) = (0.8402, 0.9438)$, and $acc_{f_{\mathbb{X}_E}}(\mathbb{T}_M) = (0.7659, 0.8811)$. Note how error table-guided sampling reaches levels of accuracy comparable to other counterexample-guided augmentation schemes (see Tab. 2) but with a third of augmenting images.

7 Conclusion

In this paper, we present a technique for augmenting machine learning (ML) data sets with counterexamples. The counterexamples we generate are synthetically-generated data items that are misclassified by the ML model. Since these items are synthesized algorithmically, their ground truth labels are also automatically generated. We show how error tables can be used to effectively guide the augmentation process. Results on training deep neural networks illustrate that our augmentation technique performs better than standard augmentation methods for image classification. Moreover, as we iterate the augmentation loop, it gets computationally harder to find counterexamples. We also show that error tables can be effective at achieving improved accuracy with a smaller data augmentation.

We note that our proposed methodology can also be extended to the use of counterexamples from "system-level" analysis and verification, where one analyzes the correctness of an overall system (e.g., an autonomous driving function) in the context of a surrounding environment [Dreossi *et al.*, 2017a]. Performing data augmentation with such "semantic counterexamples" is an interesting direction for future work [Dreossi *et al.*, 2018].

Our approach can be viewed as an instance of *counterexample-guided inductive synthesis* (CEGIS), a common paradigm in program synthesis [Solar-Lezama *et al.*, 2006; Alur *et al.*, 2013]. In our case, the program being synthesized is the ML model. CEGIS itself is a special case of *oracle-guided inductive synthesis* (OGIS) [Jha and Seshia, 2017]. For future work, it would be interesting to explore the use of oracles other than counterexample-generating oracles to augment the data set, and to compare our counterexample-guided data augmentation technique to other oracle-guided data augmentation methods.

Finally, in this work we decided to rely exclusively on simulated, synthesized data to make sure that the training, testing, and counterexample sets come from the same data source. It would be interesting to extend our augmentation method to real-world data; e.g., images of road scenes collected during driving. For this, one would need to use techniques such as domain adaptation or transfer learning [Tobin *et al.*, 2017] that can adapt the synthetically-generated data to the real world.

Acknowledgments

This work was supported in part by NSF grants 1545126, 1646208, and 1739816, the DARPA BRASS program under agreement number FA8750-16-C0043, the DARPA Assured Autonomy program, the iCyPhy center, and Berkeley Deep Drive. We acknowledge the support of NVIDIA Corporation via the donation of the Titan Xp GPU used for this research. Hisahiro (Isaac) Ito's suggestions on cross-entropy sampling are gratefully acknowledged.

References

[Alur *et al.*, 2013] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–17, October 2013.

[Ciregan *et al.*, 2012] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[Cireşan *et al.*, 2011] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[Dreossi *et al.*, 2017a] Tommaso Dreossi, Alexandre Donze, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *Proceedings of the NASA Formal Methods Conference (NFM)*, pages 357–372, May 2017.

[Dreossi *et al.*, 2017b] Tommaso Dreossi, Shromona Ghosh, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Systematic testing of convolutional neural networks for autonomous driving. In *ICML Workshop on Reliable Machine Learning in the Wild (RMLW)*, 2017. Published on Arxiv: abs/1708.03309.

[Dreossi *et al.*, 2018] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. Semantic adversarial deep learning. In *30th International Conference on Computer Aided Verification (CAV)*, 2018.

[Goodfellow *et al.*, 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Halton, 1960] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[Jha and Seshia, 2017] Susmit Jha and Sanjit A. Seshia. A Theory of Formal Synthesis via Inductive Learning. *Acta Informatica*, 54(7):693–726, 2017.

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

[Liang *et al.*, 2017] Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P Xing. Recurrent topic-transition gan for visual paragraph generation. *arXiv preprint arXiv:1703.07022*, 2017.

[Marchesi, 2017] Marco Marchesi. Megapixel size image creation using generative adversarial networks. *arXiv preprint arXiv:1706.00082*, 2017.

[Niederreiter, 1988] Harald Niederreiter. Low-discrepancy and low- sequences. *Journal of number theory*, 30(1):51–70, 1988.

[Russell *et al.*, 2015] Stuart Russell, Tom Dietterich, Eric Horvitz, Bart Selman, Francesca Rossi, Demis Hassabis, Shane Legg, Mustafa Suleyman, Dileep George, and Scott Phoenix. Letter to the editor: Research priorities for robust and beneficial artificial intelligence: An open letter. *AI Magazine*, 36(4), 2015.

[Seshia *et al.*, 2016] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. *ArXiv e-prints*, July 2016.

[Shrivastava *et al.*, 2016] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 761–769, 2016.

[Simard *et al.*, 2003] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[Sobol, 1976] Ilya M Sobol. Uniformly distributed sequences with an additional uniform property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236–242, 1976.

[Solar-Lezama *et al.*, 2006] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 404–415. ACM Press, October 2006.

[Tobin *et al.*, 2017] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Conference on Intelligent Robots and Systems, IROS*, pages 23–30. IEEE, 2017.

[van Dyk and Meng, 2001] David A van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

[Wold *et al.*, 1987] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[Wong *et al.*, 2016] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[Wu *et al.*, 2016] Bichen Wu, Forrest Iandola, Peter H. Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. 2016.

[Xu *et al.*, 2016] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. Improved relation classification by deep recurrent neural networks with data augmentation. *arXiv preprint arXiv:1601.03651*, 2016.