DEFUSE: DEBUGGING CLASSIFIERS THROUGH DISTILLING UNRESTRICTED ADVERSARIAL EXAMPLES

Anonymous authors
Paper under double-blind review

ABSTRACT

With the greater proliferation of machine learning models, the imperative of diagnosing and correcting bugs in models has become increasingly clear. As a route to better discover and fix model bugs, we propose *failure scenarios*: regions on the data manifold that are incorrectly classified by a model. We propose an end-to-end debugging framework called Defuse to use these regions for fixing faulty classifier predictions. The Defuse framework works in three steps. First, Defuse *identifies* many unrestricted adversarial examples—naturally occurring instances that are misclassified—using a generative model. Next, the procedure *distills* the misclassified data using clustering. This step both reveals sources of model error and helps facilitate labeling. Last, the method *corrects* model behavior on the distilled scenarios through an optimization based approach. We illustrate the utility of our framework on a variety of image data sets. We find that Defuse identifies and resolves concerning predictions while maintaining model generalization.

1 INTRODUCTION

The process of discovering when machine learning (ML) models fail and correcting performance in these situations is a critical part of the ML development life cycle. Uncovering bugs helps ML developers make important decisions about both development and deployment. In practice, much of debugging uses aggregate test statistics (like those in leader board style challenges [Rajpurkar et al. (2016)]) and continuous evaluation and monitoring post deployment [Liberty et al. (2020), Simon (2019)]. However, additional issues arise with over-reliance on test statistics. For instance, aggregate statistics like held out test accuracy are known to overestimate generalization performance [Recht et al. (2019)]. Further, statistics offer little insight nor remedy for specific model failures [Ribeiro et al. (2020); Wu et al. (2019)]. Last, reactive debugging of failures as they occur in production does little to mitigate harmful user experiences [La Fors et al. (2019)].

Several techniques exist for identifying undesirable behavior in machine learning models. These methods include explanations [Ribeiro et al. (2016); Slack et al. (2020b); Lakkaraju et al. (2019); Lundberg & Lee (2017)], fairness metrics [Feldman et al. (2015), Slack et al. (2020a)], data set replication [Recht et al. (2019); Engstrom et al. (2020)], and behavioral testing tools [Ribeiro et al. (2020)]. However, these techniques do not provide methods to remedy faulty model behavior or require a high level of human supervision. To better discover model bugs, we analyze *unrestricted adversarial examples*: instances on the data manifold that are misclassified [Song et al. (2018)]. We identify model failures through diagnosing common patterns in such examples.

In this work, we propose Defuse: a technique for debugging classifiers through distilling[1] unrestricted adversarial examples. Defuse works in three steps. First, Defuse *identifies* unrestricted adversarial examples by making small, semantically meaningful changes to input data using a variational autoencoder (VAE). If the classifier prediction deviates from the ground truth label on the altered instance, it returns the data instance as a potential model failure. This method employs similar techniques from [Zhao et al. (2018)]. Namely, small perturbations in the latent space of generative models can produce images that are misclassified. Second, Defuse *distills* the changes through clustering on the unrestricted adversarial example's latent codes. In this way, Defuse diagnoses regions in the latent space that are problematic for the classifier. This method produces a set of

[1] We mean distilling in the sense of "to extract the most important aspects of" and do not intend to invoke the knowledge distillation literature [Hinton et al. (2014)].

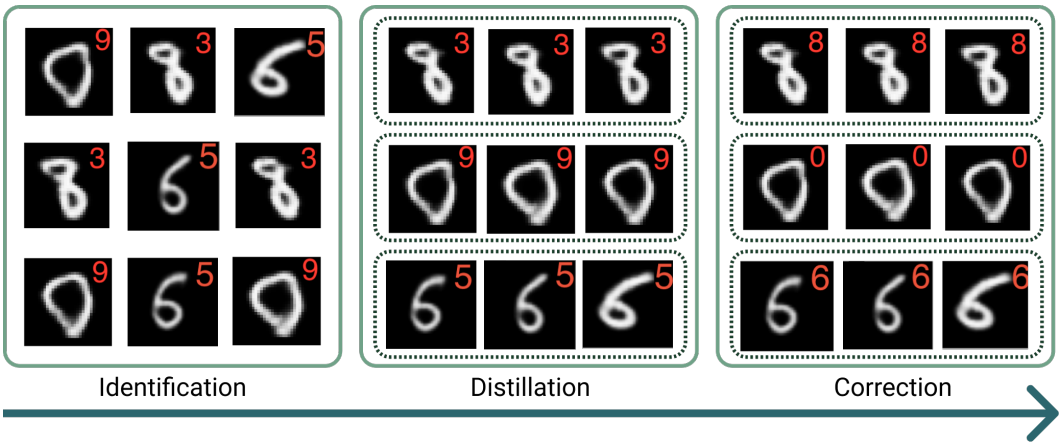


Figure 1: **Running Defuse on a MNIST classifier.** The (handpicked) images are examples from three failure scenarios identified from running Defuse. The red digit in the upper right hand corner of the image is the classifier's prediction. Defuse initially identifies many model failures. Next, it aggregates these failures in the distillation step for annotator labeling. Last, it tunes the classifier so that it correctly classifies the images, with minimal change in classifier performance. Defuse serves as an end-to-end framework to diagnose and debug errors in classifiers.

clusters in the latent space where it is likely to find misclassified data. We call these localities *failure scenarios*. An annotator reviews the failure scenarios and assigns the correct label— one label per scenario. Third, Defuse *corrects* the model behavior on the discovered failure scenarios. Because we use a generative clustering model to describe the failure scenarios, we sample many unrestricted adversarial examples and fine-tune to fix the classifier.

As an example, we run Defuse on a classifier trained on MNIST and provide an overview in figure 1. In the identification step (first pane in figure 1), Defuse generates unrestricted adversarial examples for the model. The red number in the upper right hand corner of the image is the classifier's prediction. Although the classifier achieves high test set performance, we find naturally occurring examples that are classified incorrectly. Next, the method performs the distillation step (second pane in figure 1). The clustering model groups together similar failures for annotator labeling. We see that similar mistakes are grouped together. For instance, Defuse groups together a similar style of incorrectly classified eights in the first row of the second pane in figure 1. Next, Defuse receives annotator labels for each of the clusters.[2] Last, we run the correction step using both the annotator labeled data and the original training data. We see that the model correctly classifies the images (third pane in figure 1). Importantly, the model maintains its predictive performance, scoring 99.1% accuracy after tuning. We see that Defuse enables model designers to both discover and correct naturally occurring model failures.

We first provide the necessary background and describe our method for training the VAE used in Defuse (§2). Next, we detail the three steps in Defuse: *identification*, *distillation*, and *correction* (§3). We then demonstrate the usefulness of Defuse on three image data sets: MNIST [LeCun et al. (2010)], the German traffic signs data set [Stallkamp et al. (2011)], and the Street view house numbers data set [Netzer et al. (2011)], and find that Defuse discovers and resolves critical bugs in high performance classifiers (§4).

2 Notation and Background

In this section, we establish notation and background on unrestricted adversarial examples. We also review β-VAE: the generative model we employ to find unrestricted adversarial examples. Though unrestricted adversarial examples can be found in many domains, we focus on Defuse applied to image classification.

[2]We assign label 8 to the first row in the second pane of figure 1, label 0 to the second row, and label 6 to the third row.

Unrestricted adversarial examples Let $f : \mathbb{R}^N \rightarrow [0,1]^C$ denote a classifier that accepts a data point $x \in X$, where X is the set of all legitimate images. The images consist of N features and f returns the probability that x belongs to a class $c \in \{1, ..., C\}$. Next, assume f is trained on a data set $\mathcal{D}$ consisting of d tuples (x, y) containing data point x and ground truth label y using loss function $\mathcal{L}$. Finally, suppose there exists an oracle $o : x \in X \rightarrow \{1, ..., C\}$ that outputs a label for x. Then, we define an unrestricted adversarial example to be any legitimate image where the oracle and the classifier disagree, i.e., any image belonging to the set, $\mathcal{A}_N := \{x \in X \mid o(x) \neq f(x)\}$ [Song et al. (2018)].

Variational Autoencoders (VAEs) In order to discover unrestricted adversarial examples, it is necessary to model the set of legitimate images. We use a VAE to create such a model. A VAE is composed of an encoder and a decoder neural networks. These networks are used to model the relationship between data x and latent factors $z \in \mathbb{R}^K$. Where x is generated by some ground truth latent factors $v \in \mathbb{R}^M$, we wish to train a model such that the learned generative factors closely resemble the true factors: $p(x|v) \approx p(x|z)$. In order to train such a model, we employ the β-VAE [Higgins et al. (2017)]. Specifically, we impose isotropic unit Gaussian prior over the latent factors ($p(z) = \mathcal{N}(0, I)$) and establish parametric likelihood $p_\theta(x|z)$. We train our model by minimizing the divergence between the data distribution and the model parameters with respect to θ. Namely, we optimize $\arg\min_\theta \text{KL}(p(x)||p_\theta(x)) = \arg\max_\theta \mathbb{E}_{p(x)}[\log p_\theta(x)]$ where $p_\theta(x) = \int p_\theta(x|z)p(z)dz$. However, calculating $p_\theta(x)$ is intractable, so we introduce the parametric model $q_\phi(z|x)$ and construct the variational lower bound (ELBO) with respect to $\log p_\theta(x)$, which can be shown to be:

$$L(\theta, \phi; x, z) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \, \text{KL}(q_\phi(z|x)||p(z)) \tag{1}$$

We let the approximate posterior $q_\phi(z|x)$ take on Gaussian form with diagonal covariance. Additionally, β controls the strength of the constraint on the prior. Increased β generally encourages higher degrees of disentanglement in the latent representation [Higgins et al. (2017)]. Further, $q_\phi(z|x)$ can be thought of as the encoder in a stochastic autoencoder while $p_\theta(x|z)$ is the decoder. The parameters of the neural networks defining the encoder and the decoder are ϕ and θ respectively. The parameters ϕ and θ are jointly optimized using stochastic gradient ascent.

Algorithm 1 Identification Step

procedure IDENTIFY(f, p, q, x, y, a, b)
  $\psi := \{\}$
  $\mu, \sigma := q_\phi(x)$
  for $i \in \{1, ..., Q\}$ **do**
    $\epsilon := [\text{Beta}(a, b)_1,$
      $..., \text{Beta}(a, b)_M]$
    $x_{decoded} := p_\theta(\mu + \epsilon)$
    if $y \neq f(x_{decoded})$ **then**
      $\psi := \psi \cup x_{decoded}$
    end if
  end for
  Return ψ
end procedure

Algorithm 2 Labeling Step

procedure LABEL SCENARIOS(Q, Λ, p, q, τ)
  $D_f := \{\}$
  for $(\mu, \sigma, \pi) \in \Lambda$ **do**
    $X_d := \{\}$
    for $i \in \{1, .., Q\}$ **do**
      $X_d := X_d \cup q_\theta(\mathcal{N}(\mu, \tau \cdot \sigma))$
    end for
    if $\texttt{Correct}(X_d)$ **then**
      $D_f := D_f \cup \{X_d, \texttt{Label}(X_d)\}$
    end if
  end for
  Return $\bigcup D_f$
end procedure

3 METHODS

3.1 FAILURE SCENARIOS

We begin by formalizing our notion of *failure scenarios*. Let $z \in \mathbb{R}^K$ be the latent codes corresponding to image $x \in X$. Further, let $q_\phi(\cdot) : x \rightarrow z$ be the encoder mapping the relationship between images and latent codes.

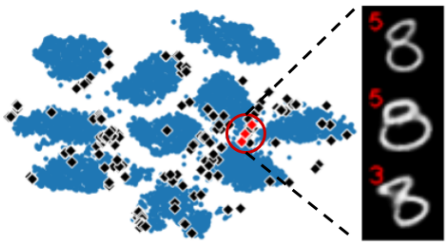


Figure 2: **Providing intuition for failure scenarios** through a t-SNE visualization of the latent space of MNIST. The black diamonds correspond to the latent codes of unrestricted adversarial examples. The blue circles are the latent codes of images from the training set. The images are three decoded latent codes (the red dots), where the red number in the right hand corner is the classifier label. We see that there are regions with higher densities of adversarial examples

Definition 3.1. Failure scenario. *Given a constant $\epsilon > 0$, matrix norm $||\cdot||$, and point z', a failure scenario is a set of images $\mathcal{A}_R = \{x \in X \mid \epsilon > ||q_\phi(x) - z'|| \wedge o(x) \neq f(x)\}$.*

Previous works that investigate unrestricted adversarial examples look for specific instances where the oracle and the model disagree [Song et al. (2018); Zhao et al. (2018)]. We instead look for regions in the latent space where this is the case. Because the latent space of the VAE tends to take on Gaussian form due to the prior, we can use euclidean distance to define these regions. If we were to define failure scenarios on the original data manifold, we may need a much more complex distance function. Because it is likely too strict to assume the oracle and model disagree on *every* instance in such a region, we also introduce a relaxation.

Definition 3.2. Relaxed failure scenario. *Given a constant $\epsilon > 0$, matrix norm $||\cdot||$, point z', and threshold ρ, a relaxed failure scenario is a set of images $\mathcal{A}_f = \{x \in X \mid \epsilon > ||q_\phi(x) - z'||\}$ such that $|\{x \in \mathcal{A}_f \mid o(x) \neq f(x)\}| \,/\, |\mathcal{A}_f| > \rho$.*

In this work, we adopt the latter definition of failure scenarios. To concretize failure scenarios and provide evidence for their existence, we continue our MNIST example from figure 1. We plot the t-SNE embeddings of the latent codes of 10000 images from the training set and 516 unrestricted adversarial examples created during the identification step in figure 2 (details of how we generate unrestricted adversarial examples in section 3.2.1). We see that the unrestricted adversarial examples are from similar regions in the latent space.

3.2 Defuse

In this section, we introduce Defuse: our procedure for identifying and correcting classifier performance on failure scenarios. First, we explain how we identity unrestricted adversarial examples using VAEs. Next, we describe our clustering approach that distills these instances into *failure scenarios*. Last, we introduce our approach to correct classifier predictions on the failure scenarios.

3.2.1 Identifying unrestricted adversarial examples

This section describes the *identification* step in Defuse (first pane in figure 1). The aim of the *identification* step is to generate many unrestricted adversarial examples. Pseudo code of the algorithm for generating a single unrestricted adversarial example is given in algorithm 1. In essence, we encode all the images from the training data. We perturb the latent codes with a small amount of noise drawn from a Beta distribution. We save instances that are classified differently from ground truth by f when decoded. We generate unrestricted adversarial examples over each instance $x \in X$ producing a set Ψ containing the ψ produced for each instance x. Where $\texttt{Identify}(\cdot)$ corresponds to algorithm 1, we define Ψ as:

$$\Psi := \bigcup \{\forall\, (x, y) \in \mathcal{D}, \texttt{Identify}(f, p, q, x, y, a, b)\} \tag{2}$$

Our technique is related to the method for generating *natural adversarial examples* from [Zhao et al. (2018)] — a very similar but slightly different concept from unrestricted adversarial examples.

The authors use a similar stochastic search method in the latent space of a GAN. They start with a small amount of noise and increase magnitude of the noise until they find a unrestricted adversarial example. Thus, they save only the unrestricted adversarial examples which are minimally distant from a data point. They also save images that differ in prediction from the original decoded instance. Because we iterate over the entire data set, it is simpler to keep the level of noise fixed and sample a predetermined number of times. In addition, we save images that differ in ground truth label from the original decoded instance because we seek to debug a classifier. Meaning, if the original instance is misclassified we wish to save this instance as a model failure.

3.2.2 DISTILLING FAILURE SCENARIOS

This section describes the *distillation* step in defuse (second pane of figure 1). The goal of the distillation step is to cluster the latent codes of the unrestricted adversarial examples (figure 2) in order to diagnose failure scenarios. We use a Gaussian mixture model with Dirichlet process prior. It is difficult to prescribe the correct number of failure scenarios ahead of the time. We can infer the correct number of failure scenarios from the data by using a non-parametric model. We define the mixture model in equation 3a.

$$p(z|\theta_1, ..., \theta_K) = \sum_{j=1}^{K} \pi_j \mathcal{N}(z|\mu_j, \Sigma_j) \quad \text{(3a)}$$

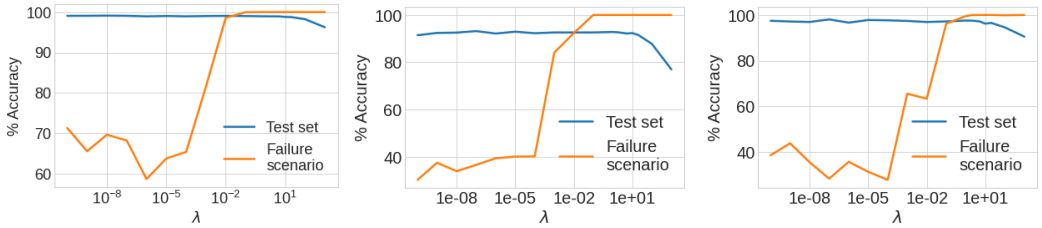
$$\begin{aligned} z_i|c_i, \Theta &\sim \mathcal{N}(\mu_{c_i}, S_{c_i}) \\ (\mu_j, S_j) &\sim G_0 \\ c_i|\pi &\sim \text{Categorical}(\pi) \\ \pi &= \text{stickbreak}(v) \\ v_K|\alpha &\sim \text{Beta}(1, \alpha) \end{aligned} \quad \text{(3b)}$$

where $\theta_j = (\mu_j, \sigma_j, \pi_j)$ is parameters for component j. This includes the mixing proportions π that sum to 1, the mean vector μ_j and precision matrix Σ_j. To make inference on our model more efficient, we use the truncated stick breaking construction of the dirchlet process, where K is the upper bound of the number of mixtures. Writing G_0 as the joint prior on the component parameters, defining indicator variable c_i where $i = 1, ..., n$, and letting stickbreak denote the truncated stick breaking distribution, we write the generative process in equation 3b. To perform inference on the model, we employ expectation maximization (EM) described in [Bishop (2006)]. In practice, we use the implementation provided in [Pedregosa et al. (2011)] (additional details provided in appendix A.1). Once we run EM and determine the parameter values, we throw away cluster components that are not used by the model. We fix some small ϵ and define the set of failure scenarios Λ generated at the distillation step as: $\Lambda := \{(\mu_j, \Sigma_j, \pi_j) | \pi_j > \epsilon\}$.

3.2.3 CORRECTING FAILURE SCENARIOS

Labeling First, an annotator assigns the correct label to the failure scenarios. For each failure scenario identified in Λ, we sample Q latent codes from $z \sim \mathcal{N}(\mu_j, \tau \cdot \sigma_j)$. Here, $\tau \in \mathbb{R}$ is a hyperparameter that controls the diversity of samples from the failure scenario. Because it could be possible for multiple ground truth classes to be present in a failure scenario, we set this parameter tight enough such that the sampled instances are from the same class to make labeling easier. We reconstruct the latent codes using the decoder $p_\theta(x|z)$. Next, an annotator reviews the reconstructed instances from the scenario and decides whether the scenario constitutes a model failure. If so, the annotator assigns the correct label to all of the instances. The correct label constitutes a single label for all of the instances generated from the scenario. We repeat this process for each of the scenarios identified in Λ. Pseudo code for the procedure is given in algorithm 2. In the algorithm, `Correct`$(\cdot)$ and `Label`$(\cdot)$ are the steps where the annotator decides whether the scenario warrants correction and uniform annotator label for each of the instances in the failure scenario.

Fine-tuning We fine-tune on the training data with an additional regularization term to fix the classifier performance on the failure scenarios. The regularization term is the cross entropy loss between the identified failure scenarios and the annotator label. Where CE is the cross entropy loss applied to the data returned from algorithm 2 and λ is the hyperparameter for the regularization term, we optimize the following objective using gradient descent: $\mathcal{F}(\mathcal{D}, \mathcal{D}_f) = \mathcal{L}(\mathcal{D}) + \lambda \cdot CE(\mathcal{D}_f)$. This objective encourages the model to maintain its predictive performance on the original training



(a) MNIST (b) SVHN (c) German Signs

Figure 3: **An overview of our results** running correction. We assess both test set accuracy and accuracy on the test failure scenario data finetuning over a range of λ's and plot the trade off. There is an optimal λ for each classifier where test set and failure scenario accuracy are both high. This result confirms that the correction step in Defuse adequately balances both these objectives .

data while encouraging the model to predict the failure instances correctly. The regularization term λ controls the pressure applied to the model to classify the failure instances correctly. For instance.

4 EXPERIMENTS

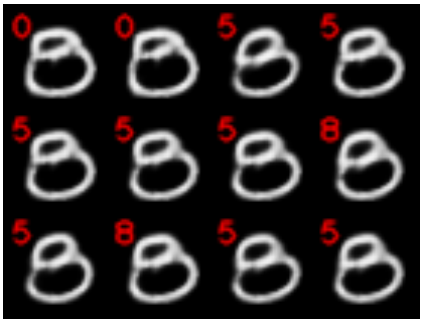
4.1 SETUP

Datasets We evaluate Defuse on three datasets: MNIST [LeCun et al. (2010)], the German Traffic Signs dataset [Stallkamp et al. (2011)], and the Street view house numbers dataset (SVHN) [Netzer et al. (2011)]. MNIST consists of $60,000$ $32x32$ handwritten digits for training and $10,000$ digits for testing. The images are labeled corresponding to the digits $0-9$. The German traffic signs data set includes $26,640$ training and $12,630$ testing images of size $128x128$. We randomly split the testing data in half to produce a validation and testing set. The images are labeled from 43 different classes to indicate the type of traffic signs. The SVHN data set consists of $73,257$ training and $26,032$ testing images. The images include digits of house numbers from Google streetview with labels $0-9$. We split the testing set in half to produce a validation and testing set.

Classifiers On MNIST, we train a CNN scoring 98.3% test set accuracy following the architecture from [Paszke et al. (2019)]. On German traffic signs and SVHN, we finetune a Resnet18 model pretrained on ImageNet [He et al. (2016)]. The German signs and SVHM models score 98.7% and 93.2% test accuracy respectively. We train a β-VAE on all avaliable data from each data set to model the set of legitimate images in Defuse. We use an Amazon EC2 P3 instance with a single NVIDIA Tesla V100 GPU for training. We follow similar architectures to [Higgins et al. (2017)]. We use the SSIM loss as the reconstruction term. We provide full training details in the appendix A.

Defuse We set K to 100 and ϵ to 0.01 during distillation. We additionally randomly down sample the number of unrestricted adversarial examples to $50,000$ to make inference of the GMM more efficient. We sample finetuning and testing sets consisting of 256 images each from every failure scenario. We swept over range of λ's and varied the number of finetuning epochs between examples. During finetuning, we select the model for each λ according to the highest training set accuracy for MNIST or validation set accuracy for SVHM and German traffic signs at the end of each finetuning epoch. Additional details are given in appendix A.

Annotator Labeling Because Defuse requires a level of human supervision, we experiment both with using Amazon Sagemaker Ground Truth to annotate failure scenarios and labeling them ourselves. When using Ground Truth, we sample 10 instances from each cluster in the distillation step. We ask 5 workers to label the instance and choose the majority class as ground truth (breaking ties randomly). We additionally let workers select "None of the above" if no label is appropriate, and we discard these instances. If more than half (i.e. setting $\rho = 0.5$) of worker labeled instances disagree with the classifier predictions on the 10 instances, we call the cluster a failure scenario. We take the majority prediction over each of the 10 ground truth labels as the label for the failure scenario. When assigning the failure scenarios ourselves, we review each cluster. If approximately half appear to be misslabeled, we call the cluster a failure scenario and assign the correct label.



(a) MNIST

(b) German traffic signs

(c) SVHN

Figure 4: **Samples from failure scenarios** in the 3 data sets. In MNIST and SVHN, we provide the predicted digit in the upper left hand corner. For German traffic signs, the label 2 corresponds to 50 km/h and 5 corresponds to 80 km/h. We find significant bugs in each of the classifiers.

4.2 DEFUSE

To evaluate Defuse we (1) assess if we identify critical failure scenarios (2) assess accuracy on the failure scenarios after correction and (3) assess the test set accuracy after correction. We measure accuracy on the failure scenarios and test set accuracy to determine if the classifier accuracy improves on the failure scenarios while maintaining generalization. We provide an overview of our results in figure 3. We provide additional details in appendix A for each experiment.

MNIST We run Defuse on MNIST using Ground Truth workers. Defuse produces 37 failure scenarios. We provide an example failure scenario in figure 4(a). Before running correction, 11.2% of the testing failure scenario images are classified according to the annotator label. We ran correction finetuning for 3 epochs over a range of λ's. In figure 3(a), we provide the trade off between test set accuracy and failure scenario accuracy. We achieve near perfect failure scenario accuracy with minimal change in test set accuracy. We provide samples from 10 additional MNIST failure scenarios in appendix C.1.

Street view house numbers We run Defuse on SVHN using Ground Truth workers. Defuse outputs 6 failure scenarios (example failure scenario in figure 4(c) additional examples in appendix C.3). The model scored 31.2% accuracy on the failure scenarios before running correction. We finetune for 5 epochs over a range of λ's. We again achieve near perfect failure scenario accuracy with minimal change in test set performance (figure 3(b))

German traffic signs Because annotating the German traffic signs data set requires more specific knowledge of the data, we run Defuse annotating the failure scenarios ourselves. Defuse produced 8 failure scenarios (example in 4(b), additional examples in appendix C.2). Before correction, the model scored 24.8% accuracy on the failure scenario test data. We finetuned for 5 epochs for a range of λ's. On German traffic signs, we achieve high failure scenario performance with minimal change to test set accuracy (figure 3(c)).

Analysis Defuse finds significant failure scenarios on each of the data sets. On MNIST, many of the 8's from one failure scenario are badly misclassified. The same is true on SVHN with regards to 5's. On German traffic signs, Defuse reveals that the classifier mistakes speed limit signs. Because of the applicability to autonomous driving, such mistakes could have significant real world impacts. These results indicate that Defuse finds critical failure scenarios. Further, Defuse either completely

fixes or improves classifier performance on the failure scenarios after correction. In addition, the models maintain their generalization performance. These results indicates that Defuse does not overfit the data from the failure scenarios. Overall, these results indicate that Defuse successfully debugs critical classifier errors.

5 Related Work

A number of related approaches for improving classifier performance use data created from generative models — mostly generative adversarial networks (GANs) [Sandfort et al. (2019); Milz et al. (2018); Antoniou et al. (2017)]. These methods use GANs to generate instances from classes that are underrepresented in the training data to improve generalization performance. Additional methods use generative models for semi-supervised learning [Kingma et al. (2014); Varma et al. (2016); Kumar et al. (2017); Dumoulin et al. (2016)]. Though these methods are similar to our work, they do not focus on identifying and presenting failures in classifiers to model designers. Also, [Varma et al. (2017)] provide a system to debug data generated from a GAN when the training set may be inaccurate. Though similar, we ultimately use a generative model to debug a classifier and do not focus on the generative model itself. Last, similar to [Song et al. (2018), Zhao et al. (2018)], [Booth et al. (2020)] provide a method to generate highly confident misclassified instances.

Related to debugging models, [Kang et al. (2018)] focus on model assertions that flag failure scenarios during production. Also, [Zhang et al. (2018)] investigate debugging the training set for incorrectly labeled instances. [Ma et al. (2018)] use state differential analysis to determine the internal features of models that produce bugs. Though these techniques are interesting approaches for debugging, we focus on preemptively identifying failures. Furthermore, we do not focus on the case of incorrectly labeled data. Also, [Ribeiro et al. (2018)] use data set augmentations which preserve the meaning of phrases in NLP for debugging models. Additionally, [Ribeiro et al. (2020)] propose a set of behavioral testing tools to find bugs in models. While useful debugging tools, these methods require a high level of supervision and expertise. Thus, they may not be useful in some situations. Also related, [Odena et al. (2019)] provide a technique to debug neural networks with a fuzzing approach. This method finds errors through perturbing data inputs with various types of noise.

6 Discussion and Conclusion

In this paper, we present Defuse: a method that generates and aggregates unrestricted adversarial examples to debug classifiers. Though unrestricted adversarial examples have been proposed in previous works, we harness such examples for the purpose of debugging classifiers. We accomplish this task through identifying failure scenarios: regions in the latent space of a VAE with many unrestricted adversarial examples. On a variety of data sets, we find that samples from failure scenarios are useful in a number of ways. First, failure scenarios are informative for understanding the ways certain models fail. Second, the generative aspect of failure scenarios is very useful for correcting failure scenarios. In our experimental results, we show that these failure scenarios include critical model issues for classifiers with real world impacts — i.e. traffic sign classification — and verify our results using ground truth annotator labels. Fortunately, we demonstrate that Defuse successfully resolves these issues.

Although Defuse identifies important errors in classifiers, the technique requires a minimal level of human supervision. Namely, the failure scenarios must be reviewed before correction. This step is necessary because some of the scenarios identified by the clustering model might have acceptable classifier performance. This could be the case because some unrestricted adversarial examples might not be found in dense latent space regions. In the future, it will be crucial to investigate automatic ways of reviewing failure scenarios. In addition, it could also be worthwhile to develop techniques with different clustering approaches. For instance, using hierarchical clustering on the unrestricted adversarial examples could produce an added level of granularity for labeling. Such approaches could lead to improved generation of failure scenarios.

REFERENCES

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *International Conference on Artificial Neural Networks and Machine Learning*, 2017.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Serena Booth, Yilun Zhou, Ankit Shah, and Julie Shah. Bayes-trex: Model transparency by example. *arXiv*, 2020.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *ICLR*, 2016.

L. Engstrom, Andrew Ilyas, Shibani Santurkar, D. Tsipras, J. Steinhardt, and A. Madry. Identifying statistical bias in dataset replication. *ICML*, 2020.

Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 259–268, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783311. URL https://doi.org/10.1145/2783258.2783311.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew M Botvinick, Shakir Mohamed, and Alexander Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning and Representation Learning Workshop*, 2014. URL http://arxiv.org/abs/1503.02531.

Daniel Kang, D. Raghavan, Peter Bailis, and M. Zaharia. Model assertions for debugging machine learning. *Debugging Machine Learning Models*, 2018.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3581–3589. Curran Associates, Inc., 2014.

Abhishek Kumar, Prasanna Sattigeri, and Tom Fletcher. Semi-supervised learning with gans: Manifold invariance with improved inference. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5534–5544. Curran Associates, Inc., 2017.

Karolina La Fors, Bart Custers, and Esther Keymolen. Reassessing values for emerging big data technologies: integrating design-based and application-based approaches. *Ethics and Information Technology*, 21(3):209–226, Sep 2019. ISSN 1572-8439. doi: 10.1007/s10676-019-09503-4. URL https://doi.org/10.1007/s10676-019-09503-4.

Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Faithful and customizable explanations of black box models. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 131–138. ACM, 2019.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, Can Balioglu, Saswata Chakravarty, Madhav Jha, Philip Gautier, David Arpin, Tim Januschowski, Valentin Flunkert, Yuyang Wang, Jan Gasthaus, Lorenzo Stella, Syama Rangapuram, David Salinas, Sebastian Schelter, and Alex Smola. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020*

ACM SIGMOD International Conference on Management of Data, SIGMOD '20, pp. 731–737, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3386126. URL https://doi.org/10.1145/3318464.3386126.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. Mode: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pp. 175–186, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355735. doi: 10.1145/3236024.3236082. URL https://doi.org/10.1145/3236024.3236082.

Stefan Milz, Tobias Rudiger, and Sebastian Suss. Aerial ganeration: Towards realistic data augmentation using conditional gans. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. volume 97 of *Proceedings of Machine Learning Research*, pp. 4901–4911, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/odena19a.html.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Mnist example pytorch. 2019. URL https://github.com/pytorch/examples.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://www.aclweb.org/anthology/D16-1264.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? volume 97 of *Proceedings of Machine Learning Research*, pp. 5389–5400, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/recht19a.html.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 856–865, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1079. URL https://www.aclweb.org/anthology/P18-1079.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.442. URL https://www.aclweb.org/anthology/2020.acl-main.442.

Veit Sandfort, Ke Yan, Perry J. Pickhardt, and Ronald M. Summers. Data augmentation using generative adversarial networks (cyclegan) to improve generalizability in ct segmentation tasks. *Scientific Reports*, 9(1):16884, Nov 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-52737-x. URL https://doi.org/10.1038/s41598-019-52737-x.

Julien Simon. Amazon sagemaker model monitor – fully managed automatic monitoring for your machine learning models. AWS News Blog, dec 2019.

Dylan Slack, Sorelle A. Friedler, and Emile Givental. Fairness warnings and fair-maml: Learning fairly with minimal data. *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*)*, 2020a.

Dylan Slack, Sophie Hilgard, Sameer Singh, and Himabindu Lakkaraju. How much should i trust you? modeling uncertainty of black box explanations. *AIES*, 2020b.

Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 8312–8323. Curran Associates, Inc., 2018.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pp. 1453–1460, 2011.

P. Varma, Bryan He, Dan Iter, Peng Xu, R. Yu, C. D. Sa, and Christopher Ré. Socratic learning: Augmenting generative models to incorporate latent subsets in training data. *arXiv: Learning*, 2016.

Paroma Varma, Dan Iter, Christopher De Sa, and Christopher Ré. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, HILDA'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350297. doi: 10.1145/3077257.3077263. URL https://doi.org/10.1145/3077257.3077263.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 747–763, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1073. URL https://www.aclweb.org/anthology/P19-1073.

Xuezhou Zhang, Xiaojin Zhu, and Stephen J. Wright. Training set debugging using trusted items. In *AAAI*, 2018.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.