Defuse: Training More Robust Models through Creation and Correction of Novel Model Errors

Dylan Slack[*]
UC Irvine
dslack@uci.edu

Krishnaram Kenthapadi
Amazon AWS
kenthk@amazon.com

Nathalie Rauschmayr
Amazon AWS
rauscn@amazon.com

Abstract

We typically compute aggregate statistics on held-out test data to assess the generalization of machine learning models. However, test data is only so comprehensive, and in practice, important cases are often missed. Thus, the performance of deployed machine learning models can be variable and untrustworthy. Motivated by these concerns, we develop methods to generate and correct novel model errors beyond those available in the data. We propose Defuse: a technique that trains a generative model on a classifier's training dataset and then uses the latent space to generate new samples which are no longer correctly predicted by the classifier. For instance, given a classifier trained on the MNIST dataset that correctly predicts a test image, Defuse then uses this image to generate new similar images by sampling from the latent space. Defuse then identifies the images that differ from the label of the original test input. Defuse enables efficient labeling of these new images, allowing users to re-train a more robust model, thus improving overall model performance. We evaluate the performance of Defuse on classifiers trained on real world datasets and find it reveals novel sources of model errors.

1 Introduction

A key goal of machine learning models is generalization. We typically measure generalization through performance on a held-out test set. Ideally, models that score well on held-out test sets should perform the same when deployed. Indeed, researchers track progress using leader boards that use such aggregate statistics [1, 2]. Nevertheless, it has become increasingly apparent test set accuracy alone does not fully describe the performance of machine learning models. For instance, statistics like held out test accuracy may overestimate generalization performance [3–5]. Also, test statistics offer little insight into or remedy specific model failures [6]. Last, test data itself is often limited and may not cover all the possible deployment scenarios [7]. Because metrics on test set data often fail to describe the performance of machine learning systems fully, it is difficult to verify and trust the behavior of machine learning models when deployed.

As a result, researchers have developed a variety of techniques to evaluate models. Such methods include explanations [8–10], fairness metrics [11, 12], and data set replication [3, 13]. Natural language processing (NLP) has increasingly turned to software engineering inspired behavioral testing tools to find errors in models [4]. Though these techniques may help find the reasons for misclassifications (e.g., explanations), they do not find novel situations in which the model fails. Other routes to discover model errors are labor-intensive and may require a high amount of task-specific expertise (e.g., dataset replication and behavioral testing).

To help remedy these issues, we introduce Defuse:[2] a technique that trains a generative model on a classifier's training dataset and then uses the latent space to generate new samples which are no

[*]Work done in part while interning at Amazon AWS.

[2]Code can be found in https://github.com/dylan-slack/Defuse

1st Workshop on eXplainable AI approaches for debugging and diagnosis (XAI4Debugging@NeurIPS2021).

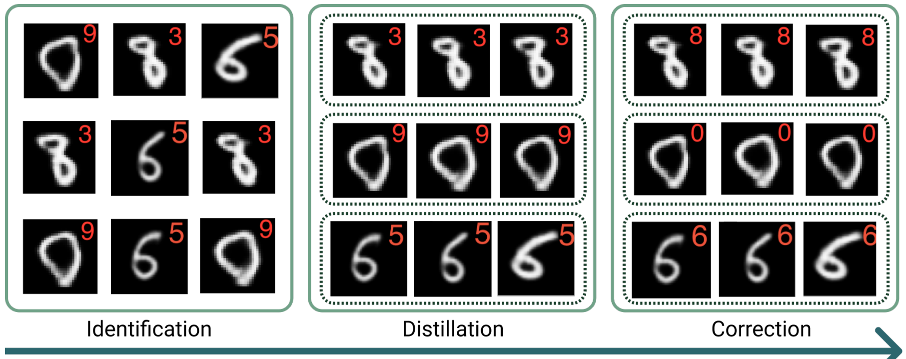


Figure 1: **Running Defuse on a MNIST classifier.** The (handpicked) images are examples from three misclassification regions identified from running Defuse. The red digit in the upper right hand corner of the image is the classifier's prediction. Defuse initially identifies many model failures. Next, it aggregates these failures in the distillation step for annotator labeling. Last, Defuse 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.

longer correctly predicted by the classifier. For instance, given a classifier trained on the MNIST dataset that correctly predicts a test image showing digit 8, Defuse then uses this image to generate new similar images by sampling from the latent space and these new images are then mistaken by the classifier for instance as digit 3. To do this, Defuse identifies regions in the latent space of the generative model associated with different types of incorrect classifier predictions and learns a secondary generative model of these regions. This process allows direct sampling of novel model misclassifications, which is useful for model understanding. We also demonstrate that similar images are usually in close proximity within the latent space and Defuse leverages this to efficiently label those mis-classifications.

For example, we run Defuse on a classifier trained on MNIST and provide an overview in figure 1. Defuse works in three steps:

1. *Identification* (first pane in figure 1): Defuse generates new images and identifies the ones that are incorrectly predicted by the classifier. By doing so, Defuse identifies regions in the latent space that are associated with errors. The red number in the upper right-hand corner of the image is the classifier's prediction. For instance, the generated images showing digit 0 are incorrectly classified as digit 9.
2. *Distillation* (second pane in figure 1): Next, the method performs the distillation step. A generative clustering model groups together the latent codes of the images from the previous step. For instance, Defuse groups together generated images showing digit 8 that are incorrectly classified as 3.
3. *Correction* (third pane in figure 1): The previous set of clusters are then annotated by labelers. Defuse then runs the correction step using both the annotator labeled generated images and the original training data. After the correction step the model correctly classifies the generated images (third pane in figure 1)

In our benchmarks, the model maintains its predictive performance, scoring 99.1% accuracy after tuning. We see that Defuse serves as a general-purpose method to both discover and correct model errors.

2 Notation and Background

In this section, we establish notation and background. Though it is conceivable to apply Defuse to many domains, we focus on images in this work.

Notation Let $f : \mathbb{R}^N \rightarrow [0, 1]^C$ denote a classifier that accepts a data point $x \in X$, where X is a set of legitimate images. The classifier f returns the probability that x belongs to 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.

Variational Autoencoders (VAEs) In order to find identified classifier mistakes, it is necessary to model the set of legitimate images X. 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 [14]. This technique produces encoder $q_\phi(z|x)$ that maps from the data and latent codes and decoder $p_\theta(x|z)$ that maps from codes to data.

3 Methods

In this section, we introduce Defuse. We describe the three main steps in the method.

3.1 Identification

Algorithm 1 Identification

Identify: Classifier f, Decoder p, Encoder q, Data point x, Label y, Beta parameters (a, b)
$\psi := \{\}$ { `Initialize misclassified images set` }
$\mu, \sigma := q_\phi(x)$ { `Compute encoded data point` }
for $i \in \{1, ..., Q\}$ **do**
  $\epsilon := [\text{Beta}(a, b)_1, ..., \text{Beta}(a, b)_M]$ { `Draw perturbations` }
  $x_{decoded} := p_\theta(\mu + \epsilon)$ { `Compute decoded perturbed data points` }
  if $y \neq f(x_{decoded})$ **then**
    $\psi := \psi \cup x_{decoded}$ { `Store the data point if its likely misclassified` }
  end if
end for
Return likely misclassified data points ψ

This section describes the *identification* step in Defuse (first pane in figure 1). The aim of the *identification* step is to generate many misclassified examples for a model. To generate these examples, we note previous works observe that slight perturbations to images in the latent space of generative models which lead to different classifications than the original image tend be misclassified [15, 16]. Following this observation, 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 use a Beta distribution so that it is possible to control the shape of the applied noise. We save instances that are classified differently from ground truth by the model f when decoded. We denote the set of these instances across all instances in a dataset as ψ. We provide pseudocode of the algorithm for generating identified classifier mistakes for a single instance x in algorithm 1.

3.2 Distillation

Next, we describe how to group the individual mistakes into high level errors. We denote these high level errors as *misclassification regions* and define them formally.

Misclassification regions Let $z \in \mathbb{R}^K$ be the latent codes corresponding to image $x \in X$ and $q_\phi(\cdot) : x \rightarrow z$ be the encoder mapping the relationship between images and latent codes.

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

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 misclassification regions on the original data manifold, we may need a much more complex distance function. Because it is likely too

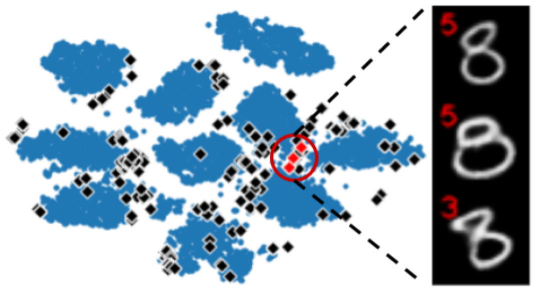


Figure 2: **Providing intuition for misclassification regions** through a t-SNE visualization of the latent space of MNIST. The black diamonds correspond to the latent codes of identified classifier mistakes. The blue circles are the latent codes of images from the training set. The images are three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

strict to assume the oracle and model disagree on *every* instance in such a region, we also introduce a relaxation.

Definition 3.2. Relaxed misclassification region. *Given a constant* $\epsilon > 0$*, vector norm* $||\cdot||$*, point* z'*, model* f*, and threshold* ρ*, a relaxed misclassification regions 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 misclassification regions. To concretize misclassification regions 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 identified classifier mistakes created during the identification step in figure 2 (details of how we generate identified classifier mistakes in section 3.1). We see that the identified classifier mistakes are from similar regions in the latent space.

3.2.1 Distilling misclassification regions

Based on our definition of misclassification regions, we describe a general procedure for learning them. We do so through clustering the latent codes of the identified classifier mistakes ψ in order to diagnose misclassification regions (second pane of figure 1). We require our clustering method to (1) infer the correct number of clusters from the data, and (2) be capable of generating instances of each cluster. We need to infer the number of clusters from the data because the number of misclassification regions is unknown ahead of time. Further, we must generate many instances from each cluster so that we have enough data to finetune on to correct the faulty model behavior. Also, generating many failure instances enables model designers to see numerous examples from the misclassification regions, which encourages understanding the model failure modes. Though any such clustering method under this description is compatible with distillation, we use a Gaussian mixture model (GMM) with the Dirichlet process prior. We use the Dirichlet process because it describes the clustering problem where the number of mixtures is unknown beforehand, fulfilling our first criteria [17]. Additionally, because the model is generative, we can sample new instances, satisfying our second criteria.

In practice, we use the truncated stick-breaking construction of the Dirichlet process, where K is the upper bound of the number of mixtures. The truncated stick-breaking construction simplifies inference making computation more efficient [17]. The method outputs a set of clusters $\theta_j = (\mu_j, \sigma_j, \pi_j)$ where $j \in \{1, ..., K\}$. The parameters μ and σ describe the mean and variance of a multivariate normal distribution and π indicates the cluster weight. To perform inference on the model, we employ expectation maximization (EM) described in [18] and use the implementation provided in [19]. 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 misclassification regions Λ generated at the distillation step as: $\Lambda := \{(\mu_j, \Sigma_j, \pi_j) | \pi_j > \epsilon\}$.

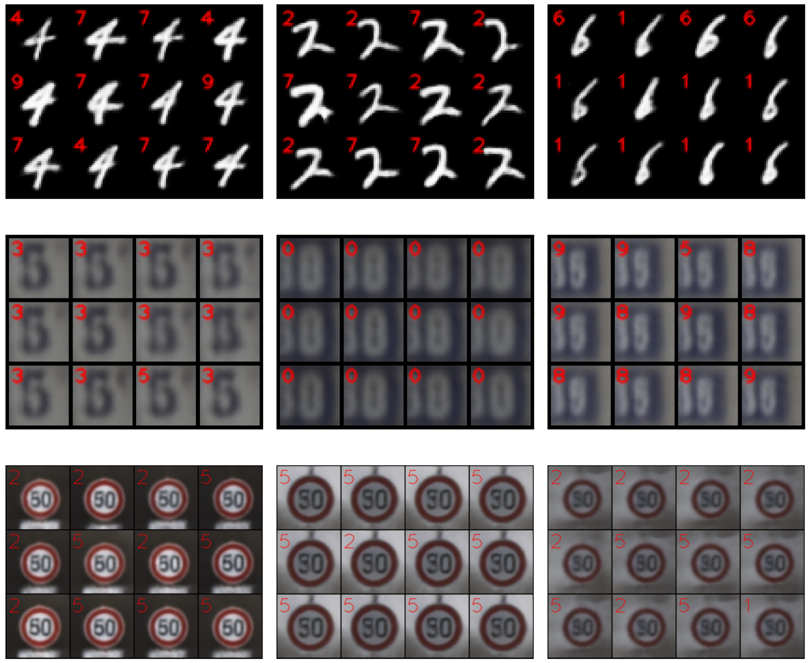


Figure 3: **Samples from three misclassification regions** from each dataset. **First row:** The MNIST misclassification regions. These scenarios were labeled 4, 2, 6 in order from left to right. **Second row:** The SVHN misclassification regions labeled 5, 8, and 5 from left to right. **Third row:** The German signs misclassification regions. The label 1 corresponds to 30km/h, 2 to 50km/h, and 5 to 80km/h. The first and second were labeled 2 while the third was labeled 1. Defuse finds significant bugs in the classifiers.

3.3 Correction

This section describes the procedure for labeling the misclassification regions and finetuning the model to fix the classifier errors.

Labeling First, an annotator assigns the correct label to the misclassification regions. For each misclassification regions 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 sample diversity from the misclassification regions. Because it could be possible for multiple ground truth classes to be present in a misclassification region, we set this parameter tight enough such that the sampled instances are from the same class. 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 and produce a dataset of failure instances $\mathcal{D}_f$. Pseudocode for the procedure is given in algorithm 3 in appendix A.

Finetuning We finetune on the training data with an additional regularization term to fix the classifier performance on the misclassification regions. The regularization term is the cross-entropy loss between the identified misclassification regions and the annotator label. Where $\mathcal{L}$ is the cross-entropy loss applied to the failure instances $\mathcal{D}_f$ and λ is the hyperparameter for the regularization term, we optimize the following objective using gradient descent, $\mathcal{F}(\mathcal{D}, \mathcal{D}_f) = \mathcal{L}(\mathcal{D}) + \lambda \cdot \mathcal{L}(\mathcal{D}_f)$ This objective encourages the model to maintain its predictive performance on the original training 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.

	Dataset	# Scenarios	Validation	Test	M.R.
Before Finetuning	MNIST	-	-	98.3	29.1
	SVHN	-	93.6	93.2	31.2
	German Signs	-	98.8	98.7	27.8
Identified Mistakes	MNIST	-	-	99.1	58.3
	SVHN	-	93.1	92.9	65.4
	German Signs	-	-	-	-
Defuse	MNIST	19	-	99.1	96.4
	SVHN	6	93.0	92.8	99.9
	German Signs	8	98.1	97.7	85.6

Figure 4: **Results from the best models** before finetuning, finetuning only on the individual mistakes, and finetuning using Defuse. The numbers presented are accuracy on the validation, test set, and misclassification region test set and the absolute number of misclassification regions generated using Defuse. We do not include finetuning on the identified mistakes for German Signs because we, the authors, assigned misclassification regions (M.R.) for this data set and thus do not have ground truth labels for individual examples. Critically, the test accuracy on the misclassification regions is high for Defuse indicating that the method successfully corrects the faulty behavior.

4 Experiments

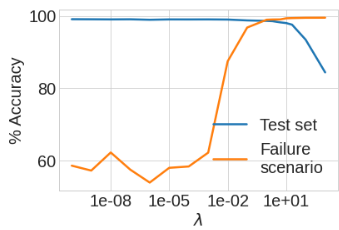
4.1 Setup

Datasets We evaluate Defuse on three datasets:[3] MNIST [20], the German Traffic Signs dataset [21], and the Street view house numbers dataset (SVHN) [22]. 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 of size 32X32. 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.

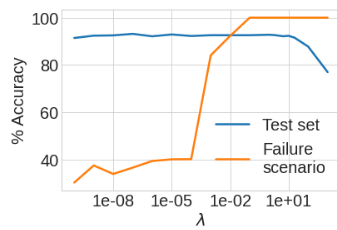
Models On MNIST, we train a CNN scoring 98.3% test set accuracy following the architecture from [23]. On German traffic signs and SVHN, we finetune a Resnet18 model pretrained on ImageNet [24]. The German signs and SVHM models score 98.7% and 93.2% test accuracy respectively. We train a β-VAE the on the training 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 [14]. We set the size of the latent dimension z to 10 for MNIST/SVHN and 15 for German signs. We provide our β-VAE architectures in appendix B.

Defuse We run Defuse on each classifier. In the identification step, we fix the parameters of the Beta distribution noise a and b to $a = b = 50.0$ for MNIST and $a = b = 75.0$ for SVHN and German signs. We found these parameters were good choices because they produce a minimal amount of perturbation noise, making the decoded instances slightly different from the original instances. During distillation, we set the upper bound on the number of components K to 100. We generally found the actual number of clusters to be much lower than this level. Thus, this serves as an appropriate upper bound. We also fixed the weight threshold for clusters ϵ to 0.01 during distillation to remove clusters with very low weighting. We also randomly downsample the number of identified classifier mistakes to $50,000$ to make the GMM more efficient. We sample finetuning and testing sets consisting of 256 images each from every misclassification region for correction. We found empirically that this number of samples is appropriate because it captures the breadth of possible images in the scenario. We use the finetuning set as the set of failure instances $\mathcal{D}_f$. We used the test set as held out data to evaluate classifier performance on the misclassification regions after correction. During sampling, we fix the sample diversity τ to 0.5 for MNIST and 0.01 for SVHN and German signs because the samples from each of the misclassification regions appear to be in the same class using these values. During correction, we finetune over a range of λ's to find the best balance between training and

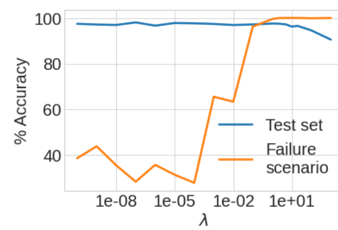
[3] Code can be found in https://github.com/dylan-slack/Defuse



(a) MNIST



(b) SVHN



(c) German Signs

Figure 5: **The tradeoff between test set and misclassification region accuracy** running correction. We assess both test set accuracy and accuracy on the test misclassification region 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 misclassification region accuracy are both high. This result confirms that the correction step in Defuse adequately balances both generalization and accuracy on the misclassification regions .

misclassification region data. We use 3 epochs for MNIST and 5 for both SVHN and German Signs because training converged within this amount of epochs. 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. We select the best model overall as the highest training or validation performance over all λ's.

Annotator Labeling Because Defuse requires human supervision, we use Amazon Sagemaker Ground Truth human workers to both determine whether clusters generated in the distillation step are misclassification regions and to generate their correct label. To determine whether clusters are misclassification regions, we sample 10 instances from each cluster in the distillation step. It is usually apparent the classifier disagrees with many of the ground truth labels within 10 instances, and thus it is appropriate to label the cluster as a misclassification region. To reduce noise in the annotation process, we assign the same image to 5 different workers and take the majority annotated label as ground truth. The workers label the images using an interface that includes a single image and the possible labels for that task. We additionally instruct workers to select "None of the above" if the image does not belong to any class and discard these labels. For instance, the MNIST interface includes a single image and buttons for the digits $0 - 9$ along with a "None of the above" button. We provide a screenshot of this interface in figure 15. 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 misclassification region. We chose $\rho = 0.5$ because clusters are highly dense with incorrect predictions at this level, making them useful for both understanding model failures and worthwhile for correction. We take the majority prediction over each of the 10 ground truth labels as the label for the misclassification region. As an exception, annotating the German traffic signs data requires specific knowledge of traffic signs. The German traffic signs data ranges across 43 different types of traffic signs. It is not reasonable to assume annotators have enough familiarity with this data and can label it accurately. For this data set, we, the authors, reviewed the distilled clusters and determined which clusters constituted misclassification regions. We labeled the clusters with more than half of the instances misclassified as misclassification regions. Though this procedure is less rigorous, the results still provide good insight into the model bugs discovered by Defuse.

4.2 Illustrative misclassification region examples

We demonstrate Defuse's potential to identify critical model bugs. We review misclassification regions from three datasets we consider. Defuse returns 19 misclassification regions for MNIST, 6 for SVHN, and 8 for German signs. We provide samples from three misclassification regions for each dataset in figure 3. The misclassification regions include numerous mislabeled examples of similar style. For example, in MNIST, the misclassification region in the upper left-hand corner of figure 3 includes a similar style of incorrectly predicted 4's. The misclassification regions generally include "corner case" images. These images are challenging to classify and thus highly insightful from a debugging perspective. For instance, the misclassified 6's are relatively thin, making them appear like 1's in some cases. There are similar trends in SVHN and German Signs. In SVHN, the model misclassifies particular types of 5's and 8's. The same is true in German signs, where the model

Misclassified Training Set

Misclassification Region

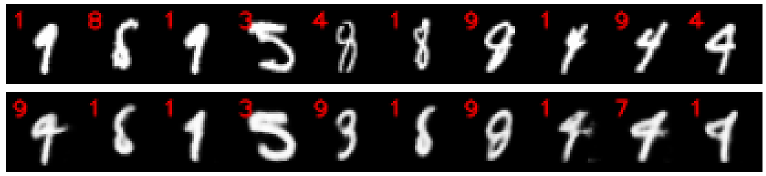


Figure 6: **Assessing the novelty of errors in misclassification regions.** We compare samples from the misclassification regions with the nearest neighbors in the misclassified training set data. We see that the misclassification regions reveal novel sources of model error not found in the misclassified training data.

predicts styles of 50km/h and 30km/h signs incorrectly. We provide additional samples from other misclassification regions in appendix D. These results demonstrate Defuse uncovers significant and insightful model bugs.

4.3 Novelty of the errors

We expect Defuse to find novel model misclassifications beyond those revealed by the available data. Thus, it is critical to evaluate whether the errors produced by Defuse are the same as those already in the training data. We compare the similarity of the errors proposed by Defuse (the misclassification region data) and the misclassified training data. We perform this analysis on MNIST. We choose 10 images from the misclassification regions and find the nearest neighbor in the misclassified training data according to ℓ_2 distance on the images. We provide the results in figure 6. We see that the data in the misclassification regions reveal different types of errors than the training set.

Interestingly, though some of the images are quite similar, they are predicted differently by the model. This result indicates the misclassification regions reveal new model failures. These results demonstrate Defuse can be used to reveal novel sources of model error.

4.4 Correcting misclassification regions

After running correction, classifier accuracy should improve on the misclassification region data indicating we have corrected the bugs discovered in earlier steps. Also, the classifier accuracy on the test set should stay at a similar level or improve, indicating that the model generalization according to the test set is still strong. We show that this is the case using Defuse. We assess accuracy on both the misclassification region test data and the original test set after performing correction. We compare Defuse against finetuning only on the images generated in the identification step that are labeled as classifier mistakes by annotators. We expect this baseline to be reasonable because related works that focus on robustness to classic adversarial attacks demonstrate the effectiveness of tuning directly on the adversarial examples [25]. We finetune on the identified classifier mistakes sweeping over a range of different λ's and taking the best model as described in section 4.1. We use this baseline for MNIST and SVHN but not German Signs because we do not have ground truth labels for identified classifier mistakes.

We provide an overview of the models before finetuning, finetuning with the identified classifier mistakes, and using Defuse in figure 4. Defuse scores highly on the misclassification region data after correction compared to before finetuning. There is only a marginal improvement in the baseline. These results indicate Defuse corrects the faulty model performance on the misclassification regions. Also, these results show the clustering step in Defuse is critical to its success. We see this because finetuning on the identified classifier mistakes performs worse than finetuning on the misclassification regions. Last, there are minor effects on test set performance during finetuning, demonstrating Defuse does not harm generalization according to the test set.

Further, we plot the relationship between test set accuracy and misclassification region test accuracy in figure 5 varying over λ. There is an appropriate λ for each model where test set accuracy and accuracy on the misclassification regions are both high. Overall, these results indicate the correction step in Defuse is highly effective at correcting the errors discovered during identification and distillation.

Dataset	M.R.	Non-M.R.	Combined
MNIST	$78.9.3 \pm 5.4$	87.2 ± 3.2	85.2 ± 0.1
SVHN	66.6 ± 8.4	83.2 ± 4.1	82.1 ± 1.3

Figure 7: **Annotator agreement** on the identified classifier mistakes. We plot the mean and standard error of the percent of annotators that voted for the majority label in an identified classifier mistakes across all the annotated examples. We break this down into the misclassification region (M.R.) and non-misclassification region (Non-M.R.) identified classifier mistakes and the combination between the two. The annotators are generally in agreement though less so for the misclassification region data, indicating these tend to be more ambiguous examples.

4.5 Annotator Agreement

Because we rely on annotators to provide the ground truth labels for the identified classifier mistakes, we investigate the agreement between the annotators during labeling. The annotators should agree on the labels for the identified classifier mistakes. Agreement indicates we have high confidence our evaluation is based on accurately labeled data. We evaluate the annotator agreement by assessing the percent of annotators that voted for the majority label prediction for a single instance. This metric will be high when the annotators agree and low when only a few annotators constitute the majority vote. Further, we calculate the annotator agreement for every annotated instance. We provide the annotator agreement on MNIST and SVHN in figure 7 broken down into misclassification region data, non-misclassification region data, and their combination. Interestingly, the misclassification region data has slightly lower annotator agreement indicating these tend to be more ambiguous examples. Further, there is lower agreement on SVHN than MNIST, likely because this data is more complex. All in all, there is generally high annotator agreement across all the data.

5 Related Work

A number of related approaches for improving classifier performance use data created from generative models — mostly generative adversarial networks (GANs) [26–29]. 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 [30–33]. Though these methods are similar in nature to the correction step of our work, a key difference is Defuse focuses on summarizing and presenting high level model failures. Also, [34] 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 [16], [15], [35] provide a method to generate highly confident misclassified instances.

One aspect of our work looks at improving performance on identified classifier mistakes. Thus, there are similarities between our work and methods that improve robustness to adversarial attacks. Similar to Defuse, several techniques demonstrate that tuning on additional data helps improve classic adversarial robustness. [36] demonstrate robustness is improved with the addition of unlabeled data during training. Also, [37] show directly training on the adversarial examples improves robustness. [38] characterize a trade-off between robustness and accuracy in perturbation based data augmentations during adversarial training. Because we train with on manifold data created by generative models, there is not so much of a trade-off between robustness and accuracy. We find that we can achieve high performance on the identified classifier mistakes with minimal change to test accuracy. Though related, our work demonstrates that robustness to naturally occurring adversarial examples show different robustness dynamics than classic adversarial examples.

Related to debugging models, [39] focus on model assertions that flag failures during production. Also, [37, 40] investigate debugging the training set for incorrectly labeled instances. We focus on preemptively identifying model bugs and do not focus on incorrectly labeled test set instances. Additionally, [4] propose a set of behavioral testing tools that help model designers find bugs in NLP models. This technique requires a high level of supervision and thus might not be appropraite in some settings. Last, [41] provide a technique to debug neural networks through perturbing data inputs with various types of noise.

6 Conclusion

In this paper, we present Defuse: a method that summarizes and corrects high level model errors. Our experimental results show that Defuse identifies critical model issues for classifiers with real world impacts (i.e. traffic sign classification) and verify our results using ground truth annotator labels. We also demonstrate that Defuse successfully resolves these issues. A potential direction for future research is to explore directly optimizing for the misclassification regions, instead of identifying individual example and clustering.

7 Acknowledgments

This work was supported by Amazon AWS. DS would also like to acknowledge support from the HPI Research Center in Machine Learning and Data Science at UC Irvine.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[2] 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*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://www.aclweb.org/anthology/D16-1264.

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

[4] 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*, pages 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.

[5] Kayur Patel, James Fogarty, James A. Landay, and Beverly Harrison. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 667–676, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580111. doi: 10.1145/1357054.1357160. URL https://doi.org/10.1145/1357054.1357160.

[6] 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*, pages 747–763, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1073. URL https://www.aclweb.org/anthology/P19-1073.

[7] Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.117. URL https://www.aclweb.org/anthology/2020.findings-emnlp.117.

[8] 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*, pages 1135–1144, 2016.

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

[10] 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, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[11] 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, page 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.

[12] Disi Ji, Padhraic Smyth, and Mark Steyvers. Can i trust my fairness metric? Assessing fairness with unlabeled data and Bayesian inference. In *Advances in Neural Information Processing Systems*, 2020.

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

[14] 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.

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

[16] 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, editors, *Advances in Neural Information Processing Systems 31*, pages 8312–8323. Curran Associates, Inc., 2018.

[17] Erik B. Sudderth. Graphical models for visual object recognition and tracking. *PhD Thesis, MIT*, 2006.

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

[19] 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.

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

[21] 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*, pages 1453–1460, 2011.

[22] 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.

[23] 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.

[24] 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)*, pages 770–778, 2016.

[25] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. volume 97 of *Proceedings of Machine Learning Research*, pages 7472–7482, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/zhang19p.html.

[26] 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.

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

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

[29] Oran Lang, Yossi Gandelsman, Michal Yarom, Yoav Wald, Gal Elidan, Avinatan Hassidim, William T. Freeman, Phillip Isola, Amir Globerson, Michal Irani, and Inbar Mosseri. Explaining in style: Training a gan to explain a classifier in stylespace. *ICCV*, 2021.

[30] 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, editors, *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc., 2014.

[31] 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.

[32] 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, editors, *Advances in Neural Information Processing Systems 30*, pages 5534–5544. Curran Associates, Inc., 2017.

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

[34] 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.

[35] Serena Booth, Yilun Zhou, Ankit Shah, and Julie Shah. Bayes-TrEx: Model transparency by example. In *AAAI*, 2021.

[36] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John Duchi. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

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

[38] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John Duchi, and Percy Liang. Understanding and mitigating the tradeoff between robustness and accuracy. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7909–7919. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/raghunathan20a.html.

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

[40] Alex Bäuerle, Heiko Neumann, and Timo Ropinski. Training de-confusion: An interactive, network-supported visual analysis system for resolving errors in image classification training data. 08 2018.

[41] 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*, pages 4901–4911, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/odena19a.html.

A Defuse Psuedo Code

In algorithm 3, `Correct`$(\cdot)$ and `Label`$(\cdot)$ are the steps where the annotator decides if the scenario warrants correction and the annotator label for the misclassification region.

Algorithm 2 Identification

Identify: f, p, q, x, y, a, b
$\psi := \{\}$
$\mu, \sigma := q_\phi(x)$
for $i \in \{1, ..., Q\}$ **do**
  $\epsilon :=$ [Beta$(a, b)_1$,
    ..., 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

Algorithm 3 Labeling

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

B Training details

B.1 GMM details

In all experiments, we use the implementation of Gaussian mixture model with dirichlet process prior from [[19]]. We run our experiments with the default parameters and full component covariance.

B.2 MNIST details

Model details We train a CNN on the MNIST data set using the architecture in figure 8. We used the Adadelta optimizer with the learning rate set to 1. We trained for 5 epochs with a batch size of 64.

Architecture
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 10 $\times$ 2

Figure 8: MNIST CNN Architecture

β-VAE training details We train a β-VAE on MNIST using the architectures in figure 9 and 10. We set β to 4. We trained for 800 epochs using the Adam optimizer with a learning rate of 0.001, a minibatch size of 2048, and β set to 0.4. We also applied a linear annealing schedule on the KL-Divergence for 500 optimization steps. We set z to have 10 dimensions.

Architecture
4x4 conv., 32 ReLU stride 2
4x4 conv., 32 ReLU stride 2
4x4 conv., 32 ReLU stride 2
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 15 $\times$ 2

Figure 9: MNIST data set encoder architecture.

Architecture
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 256, ReLU
4x4 transpose conv., 32 ReLU stride 2
4x4 transpose conv., 32 ReLU stride 2
4x4 transpose conv., 32 ReLU stride 2
4x4 transpose conv., 32 Sigmoid stride 2

Figure 10: MNIST data set decoder architecture.

Identification We performed identification with Q set to 500. We set a and b both to 50. We ran identification over the entire training set. Last, we limited the max allowable size of ψ to 100.

Distillation We ran the distillation step setting K, the upper bound on the number of mixtures, to 100. We fixed ϵ to 0.01 and discarded clusters with mixing proportions less than this value. This left 44 possible scenarios. We set τ to 0.5 during review. We used Amazon Sagemaker Ground Truth to determine misclassification regions and labels. The labeling procedure is described in section 4.1. This produced 19 misclassification regions.

Correction We sampled 256 images from each of the misclassification regions for both finetuning and testing. We finetuned with minibatch size of 256, the Adam optimizer, and learning rate set to 0.001. We swept over a range of correction regularization λ's consisting of $[1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 2, 5, 10, 20, 100, 1000]$ and finetuned for 3 epochs on each.

B.3 German Signs Dataset Details

Dataset The data consists of 26640 training images and 12630 testing images consisting of 43 different types of traffic signs. We randomly split the testing data in half to produce 6315 testing and validation images. Additionally, we resize the images to 128x128 pixels.

Classifier f We fine-tuned the ResNet18 model for 20 epochs using Adam with the cross entropy loss, learning rate of 0.001, batch size of 256 on the training data set, and assessed the validation accuracy at the end of each epoch. We saved the model with the highest validation accuracy.

β-VAE training details We trained for 800 epochs using the Adam optimizer with a learning rate of 0.001, a minibatch size of 2048, and β set to 4. We also applied a linear annealing schedule on the KL-Divergence for 500 optimization steps. We set z to have 15 dimensions.

Architecture
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 15×2

Figure 11: German signs data set encoder architecture.

Identification We performed identification with Q set to 100. We set a and b both to 75.

Distillation We ran the distillation step setting K to 100. We fixed ϵ to 0.01 and discarded clusters with mixing proportions less than this value. This left 38 possible scenarios. We set τ to 0.01 during review. We determined 8 of these scenarios were particularly concerning.

Architecture
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 256, ReLU
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 Sigmoid stride 2

Figure 12: German signs data set decoder architecture.

Correction We finetuned with minibatch size of 256, the Adam optimizer, and learning rate set to 0.001. We swept over a range of correction regularization λ's consisting of $[1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 2, 5, 10, 20, 100, 1000]$ and finetuned for 5 epochs on each.

B.4 SVHN details

Dataset The data set consists of 73257 training and 26032 testing images. We also randomly split the testing data to create a validation data set. Thus, the final validation and testing set correspond to 13016 images each.

Classifier f We fine tuned for 10 epochs using the Adam optimizer, learning rate set to 0.001, and a batch size of 2048. We chose the model which scored the best validation accuracy when measured at the end of each epoch.

β**-VAE training details** We trained the β-VAE for 400 epochs using the Adam optimizer, learning rate 0.001, and minibatch size of 2048. We set β to 4 and applied a linear annealing schedule on the Kl-Divergence for 5000 optimization steps. We set z to have 10 dimensions.

Architecture
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
4x4 conv., 64 ReLU stride 2
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 10×2

Figure 13: SVHN data set encoder architecture.

Architecture
Fully connected 256, ReLU
Fully connected 256, ReLU
Fully connected 256, ReLU
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 ReLU stride 2
4x4 transpose conv., 64 Sigmoid stride 2

Figure 14: SVHN data set decoder architecture.

Identification We set Q to 100. We also set the maximum size of ψ to 10. We set a and b to 75.

Distillation We set K to 100. We fixed ϵ to 0.01. The distillation step identified 32 plausible misclassification regions. The annotators deemed 6 of these to be misclassification regions. We set τ to 0.01 during review.

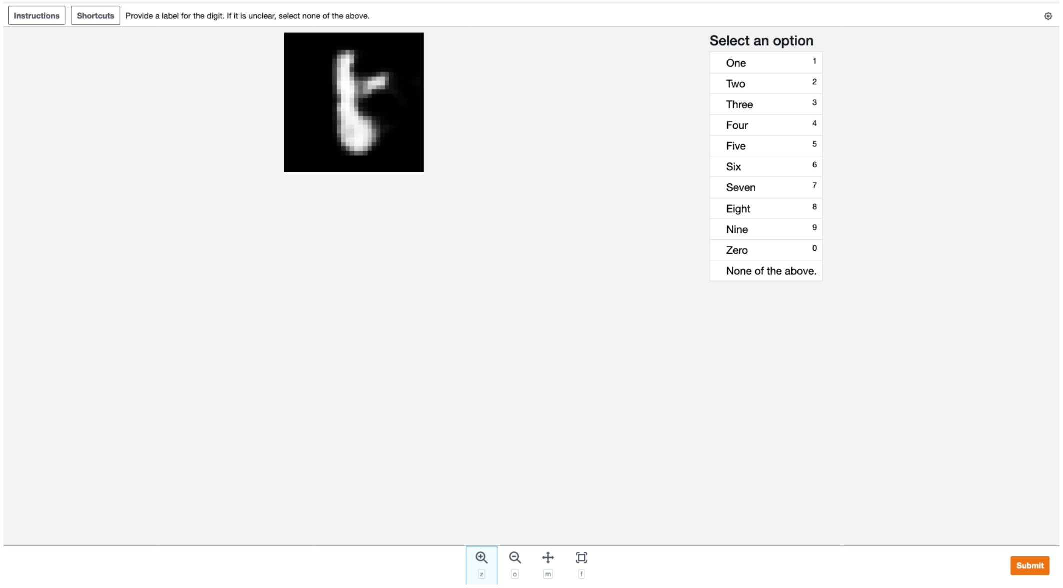


Figure 15: Annotation interface.

Correction We set the minibatch size of 2048, the Adam optimizer, and learning rate set to 0.001. We considered a range of λ's: $[1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 2, 5, 10, 20, 100, 1000]$. We finetuned for 5 epochs.

B.5 t-SNE example details

We run t-SNE on $10,000$ examples from the training data and 516 identified classifier mistakes setting perplexity to 30. For the sake of clarity, we do not include outliers from the identified classifier mistakes. Namely, we only include identified classifier mistakes with $> 1\%$ probabilitdefuse-iclr clusters.

C Annotator interface

We provide a screenshot of the annotator interface in figure 15.

D Additional experimental results

D.1 Additional samples from MNIST misclassification regions

We provide additional examples from 10 randomly selected (no cherry picking) MNIST misclassification regions. We include the annotator consensus label for each misclassification region.

Figure 16: Annotator label 6.

Figure 17: Annotator label 3.

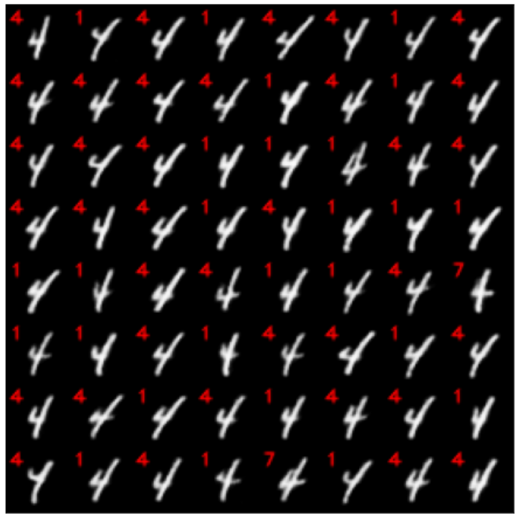


Figure 18: Annotator label 4.

Figure 19: Annotator label 4.

Figure 20: Annotator label 6.

Figure 21: Annotator label 8.

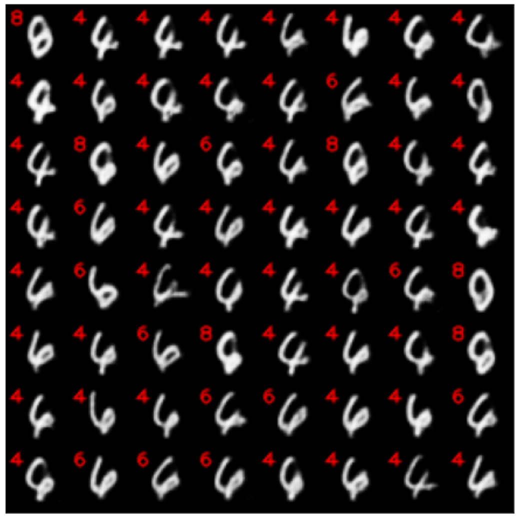


Figure 22: Annotator label 6.

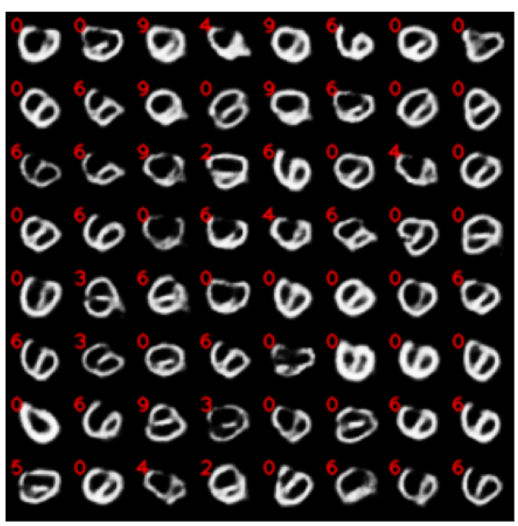


Figure 23: Annotator label 0.

Figure 24: Annotator label 6.

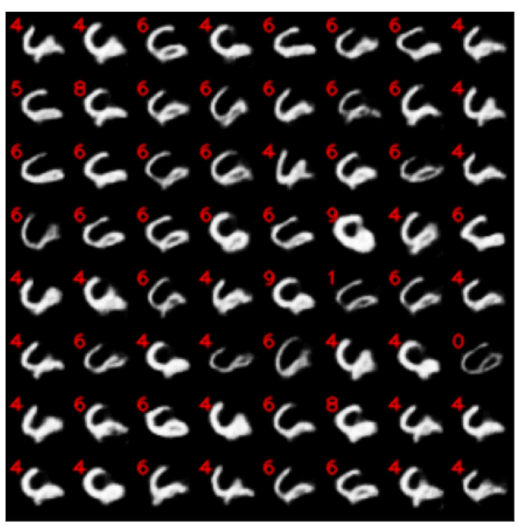


Figure 25: Annotator label 6.

D.2 Additional samples from German signs misclassification regions

We provide samples from all of the German signs misclassification regions. We provide the names of the class labels in figure 26. For each misclassification region, we indicate our assigned class label in the caption and the classifier predictions in the upper right hand corner of the image.

ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric tons

Figure 26: German signs class labels.

Figure 27: Annotator label 7.

Figure 28: Annotator label 2.

Figure 29: Annotator label 7.

Figure 30: Annotator label 41.

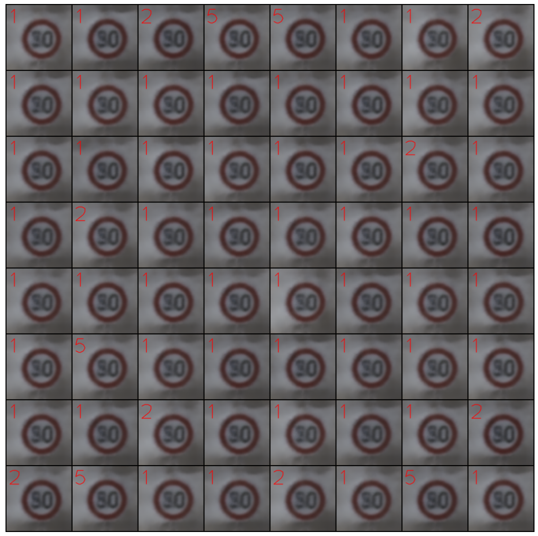


Figure 31: Annotator label 1.

Figure 32: Annotator label 2.

Figure 33: Annotator label 2.

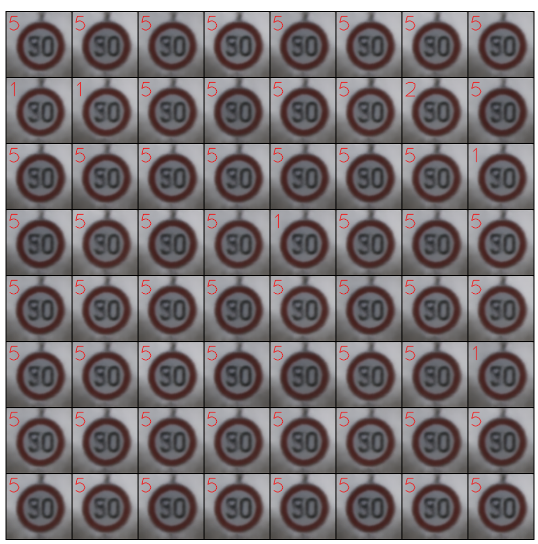


Figure 34: Annotator label 1.

Figure 35: Annotator label 2.

D.3 Additional samples from SVHN misclassification regions

We provide additional samples from each of the SVHN misclassification regions. The digit in the upper left hand corner is the classifier predicted label. The caption includes the Ground Truth worker labels.

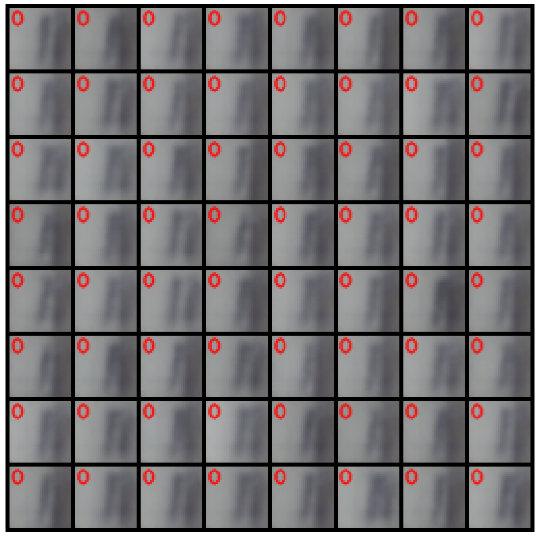


Figure 36: Annotator label 1.

Figure 37: Annotator label 5.

Figure 38: Annotator label 8.

Figure 39: Annotator label 0.

Figure 40: Annotator label 3.

Figure 41: Annotator label 5.