Adversarial Weakness Recognition for Efficient Black-Box Validation

Efficiently select candidate failures to avoid exhaustive validation

Robert J. Moss
Computer Science
Stanford University
`mossr@cs.stanford.edu`

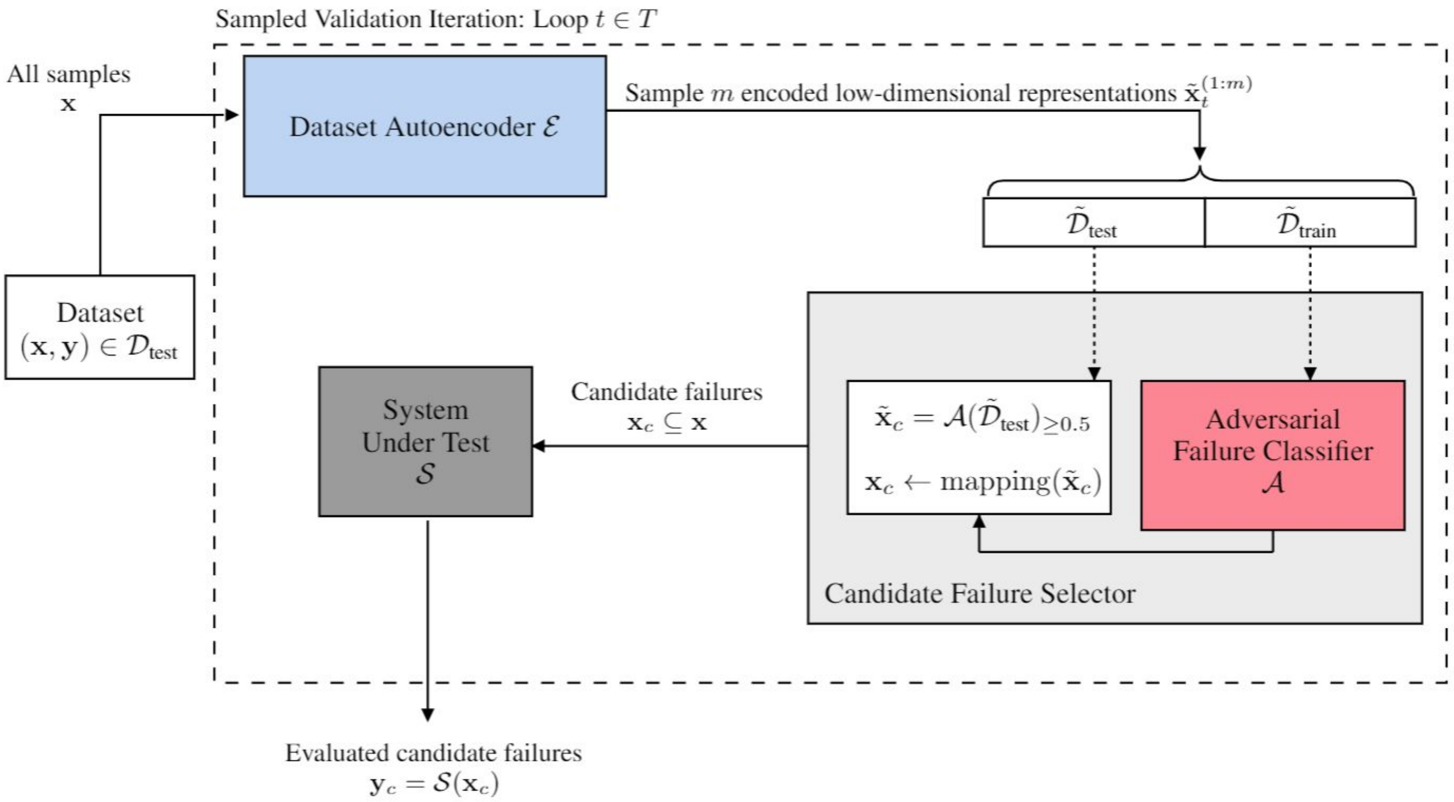
Motivation: Black-box validation

1. **Validate a black-box system without exhaustively iterating the entire validation set**
    - Want to focus on failure cases

2. **Automate the process of selecting failures**
    - Do not want to hand pick failures, or randomly sample candidate inputs to test

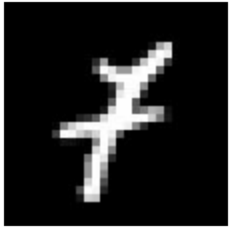
3. **Intended for systems that are computationally expensive to call**
    - Also intended for very large datasets that may be computationally intractable to exhaustively evaluate

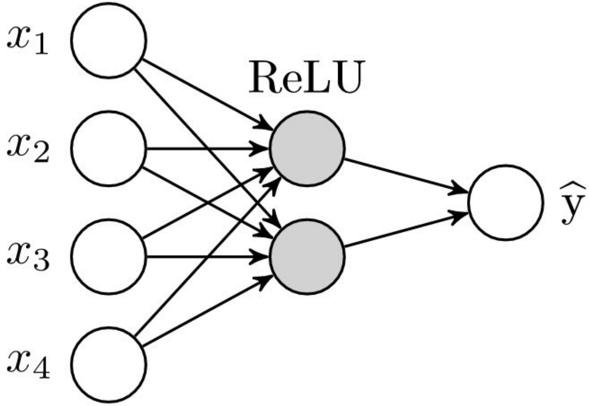
Method: Validation Framework



Black-Box System Under Test: MNIST classifier

- **MNIST classifier trained on 60,000 28x28 gray-scale images**
  - Used the Julia machine learning package `Flux.jl` for modelling and training
  - Two dense layers with a ReLU activation
  - Trained using the logit cross-entropy loss function: $\mathcal{L}_S(\mathrm{softmax}(\hat{\mathbf{y}}), \mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i)$
- **Treated as a "black-box" system**
  - We only care about passing inputs and parsing outputs
- **Achieves about 93.2% accuracy, so known failures exist**
  - Failure is defined as a *misclassification*
  - Example: misclassified this 7 incorrectly as a 1

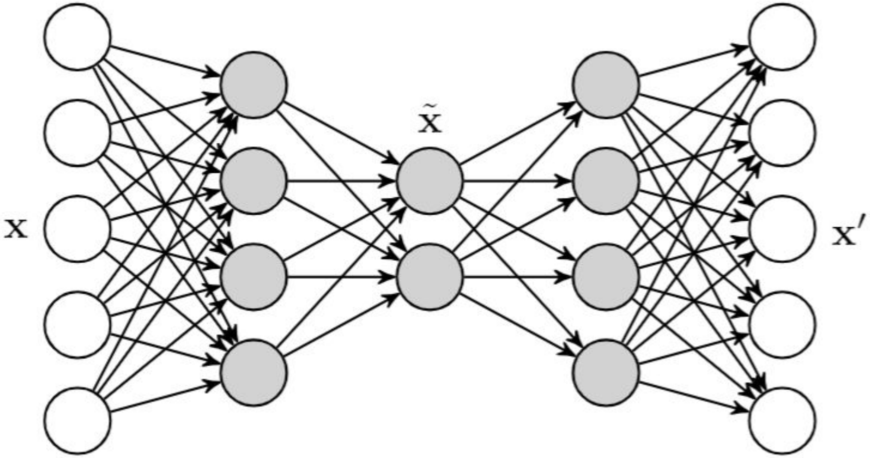




Dataset Autoencoder: Low-dimensional representation

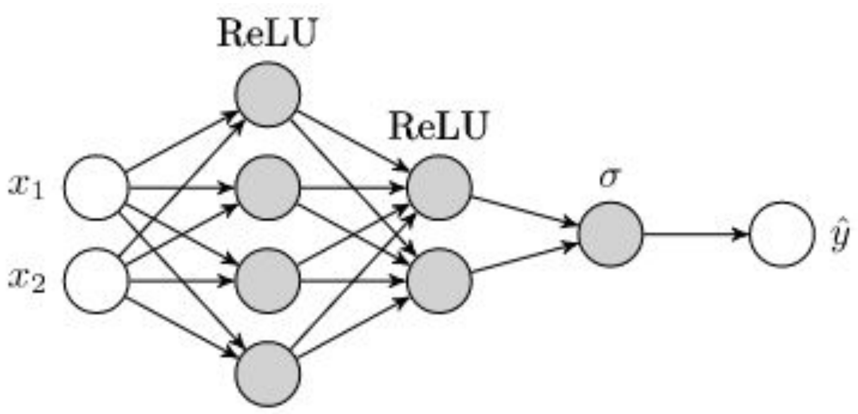
- **Represent dataset inputs x as a low-dimensional vector (trained on MNIST *test* dataset)**
  1. Reduces the input size
  2. We can learn low-dimensional feature representations that led to failures
- **Trained as a unsupervised autoencoder network, using LeakyReLU activations**
  - We use the mean squared error loss:

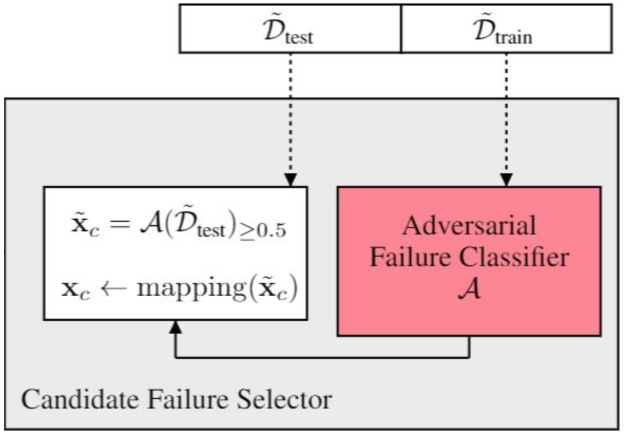
$$\mathcal{L}_{\mathcal{E}}(\mathbf{x}', \mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m}(x_i' - x_i)^2$$



Adversarial Failure Classifier: Select failures

- **Adversary is framed as a failure classifier, trained on the portion of low-dimensional samples**
  - Trained using the binary cross-entropy loss function
- **Failure prediction is output after being passed through a sigmoid layer to turn it into a probability**

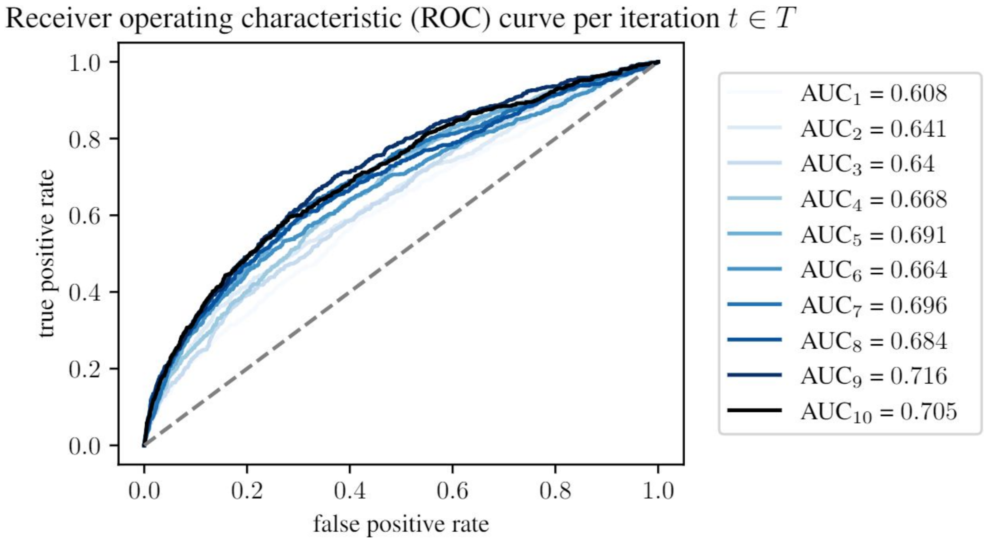


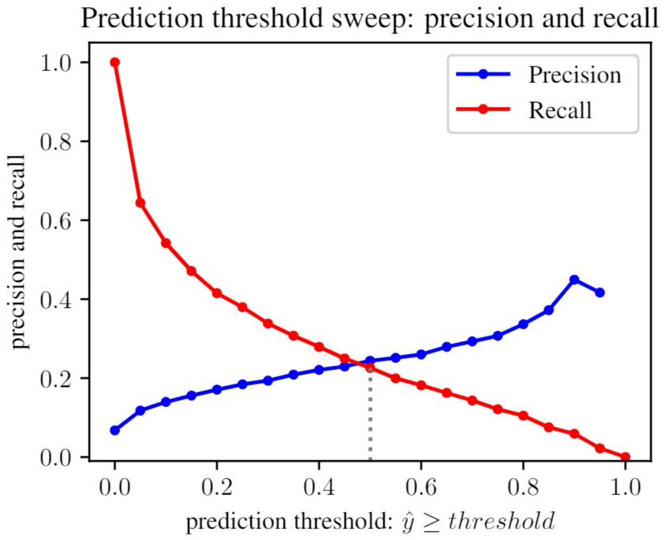


$$\mathcal{L}_{\mathcal{A}}(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

Experiments: ROC and AUC

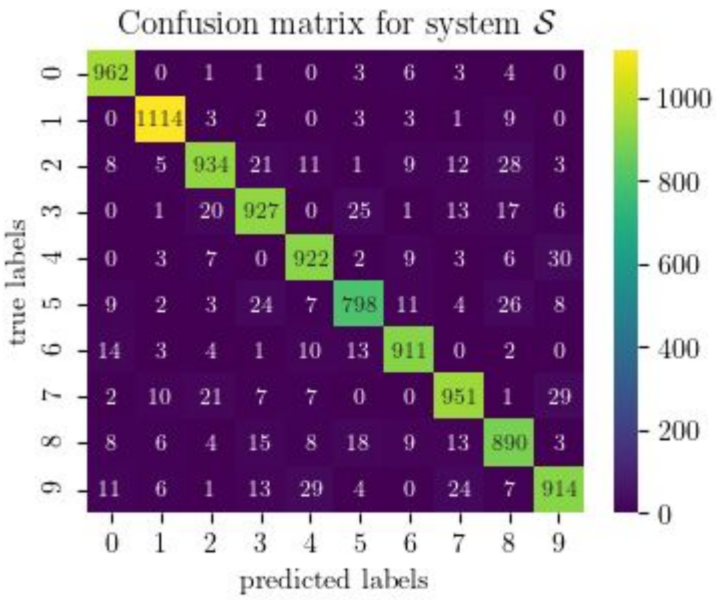
- **To assess the false positive rate and true positive rate, ROC curves and their AUC were calculated**
  - We also swept the prediction threshold to balance the trade-off between *precision* and *recall*

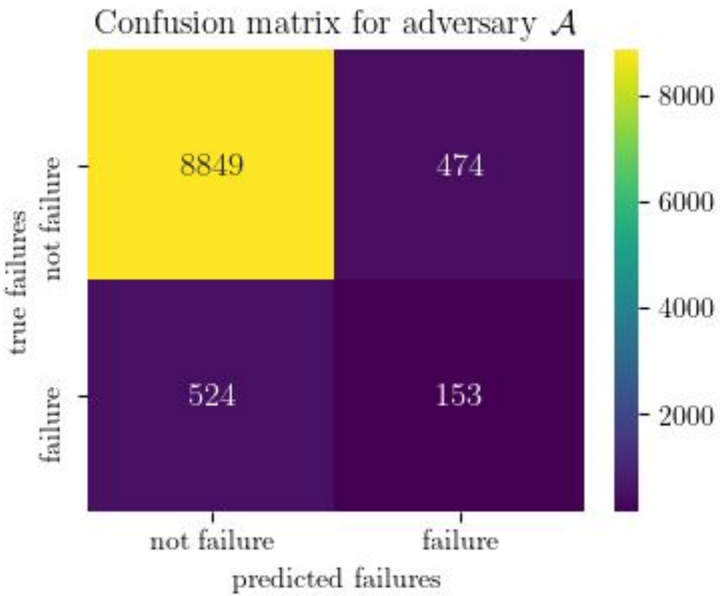




Experiments: Confusion matrices

- **Confusion matrices for the classification system under test $\mathscr{S}$ and adversarial failure classifier $\mathscr{A}$**
  - System under test predicts well, which makes it tough for the adversary to predict rare failures





Results and Analysis: Evaluation metrics

- **Metrics of *precision* and *recall* were used to evaluate the adversary against a random selector**
    - During the sampled validation iteration loop (i.e. the framework loop), the adversary will select failures at a rate 3 times more likely than random
    - The system has a true failure rate of about 0.0677, which roughly matches the precision of *random*
    - Precision for the adversary is much higher, given that the rate of true failures is so low
    - The balance of precision and recall is also good for the adversary

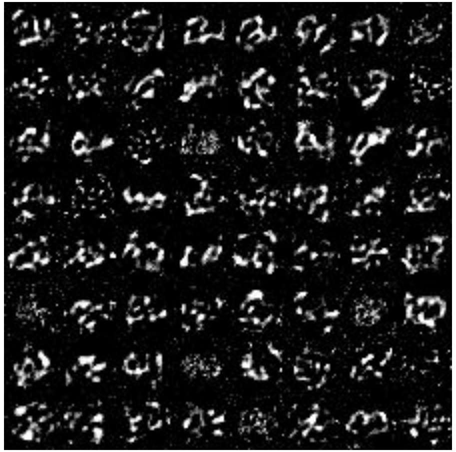
Table 1: Evaluation Metrics

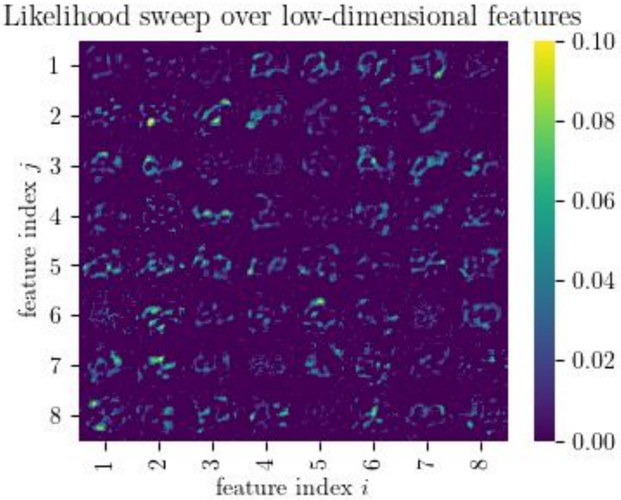
Failure Selector	**Precision***	**Recall***	**Sampled Precision†**	**Sampled Recall†**
Adversary $\mathcal{A}$	0.2441	0.2260	0.2374 ± 0.11	0.3244 ± 0.17
Random	0.0647	0.4712	0.0618 ± 0.04	0.0910 ± 0.07

* Run over $\mathcal{D}_{\text{test}}$ only calculated for the "failure" class.
† Calculated from $T = 10$ iterations of the *sampled validation loop*.

Results and Analysis: Failure feature space

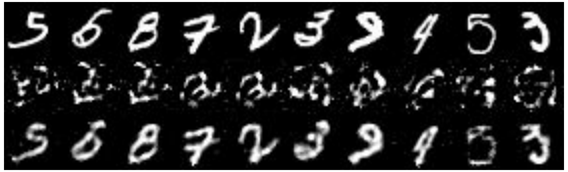
- **We took 64 one-hot vectors, each of size 64, and "activated" each individual index then decoded**
  - **Left**: this shows a mapping of the low-dimensional (64) encoded space back to images (28x28)
  - **Right**: we then use the adversary to generate a likelihood value given each of these one-hot vectors
    - This indicates the areas of the features that are likely to cause failures





Results and Analysis: Analyzing predictions

- **Analyzing predictions made by the adversary**
  - We show *true positives* and *false negatives*, noticing similar feature representations (middle row)
  - We also show the highest likely predicted failures (with indications whether they're true failures or not)



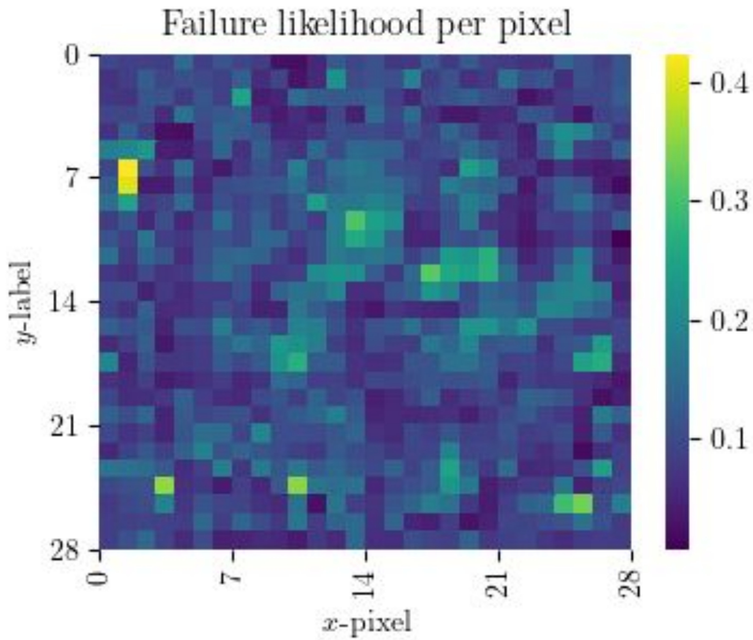
true positives

false negatives

likely failures

Visualizations: Failure likelihood per pixel

- **To see which element of the low-dimensional feature vector contributed the most to a likely failure as predicted by the adversary, we encode a one-hot vector over $\mathbb{R}^{28x28}$ and plot the likelihood per pixel**



Conclusions: Discussion and future work

- **We propose a validation framework for iteratively selecting candidate failures**
  - With the intention to reduce computational load on the system under test
- **Use an autoencoder to get a low-dimensional representation on the inputs**
  - Learn which low-dimensional features are likely to result in a failure
- **A supervised adversarial failure classifier selects candidate inputs that are likely to be system failure**
  - Selects failures 3 times more likely than random—especially important for a system with rare failures
- **Future work would extend this framework into a continual learning domain**
  - How can we learn from these failures to improve the system under test?
  - Then, can we use past known failures to look for failures in the updated system?