**Motivation**

Deep learning is rapidly transitioning to be a dominant machine learning technique in multiple fields including data analytics, autonomous systems, and security. While deep learning has enabled us to tackle a variety of unique problems, Papernot et. al. have shown that such networks are vulnerable to adversarial examples: malicious inputs crafted by an adversary from applying slight perturbations to legitimate inputs. These adversarial examples pose new hurdles for many subfields, including artificial intelligence & information security.

In this work, we use techniques from other fields in computer science to extract high-level, domain-specific, and “human-inspired” features to detect the presence of adversarial examples in real-time with a couple of simple checks. We call these high-level features embedded semantics.

**Crafting Adversarial Examples**

To evaluate the impact of pairing a deep neural network with an external verifier, we first generate adversarial examples from legitimate inputs using the Jacobian Saliency Map Approach (Papernot et. al.). In this approach, we compute the Jacobian Matrix of the network’s learned function. This matrix describes how changes to inputs will vary the outputs of the neural network. Finally, input components are selected by creating saliency maps, which are maps defined using formulae based on the Jacobian matrix with the adversarial goal in mind.

After the desired input components are identified, slight perturbations are added/removed to/from the inputs, and the process repeats until the network has misclassified the sample.

**Technical Approach - Overview**

After generating our adversarial examples, we borrow techniques from the computer vision domain to perform an independent classification. Our classification is derived from contour comparisons, prominent color identification, and vertex enumeration.

To determine the shape of our input, we first slightly blur the image to remove noise, apply a threshold to create a binary image, and extract the largest contour under a maximum area constraint.

Next, we measure the similarity between the selected contour and contours defined from characteristic shapes of signs (triangles, circles, diamonds, octagons, etc.). Finally, we further filter our list of possible classes based on prominent color, number of vertices, or orientation.

**Results - Accuracy**

In this work, we trained our network and tailored our detector to the German Traffic Sign Recognition Benchmark (GTSRB), which contains ~50,000 images of different traffic signs spanning over 40 unique classes.

We generated over 250 adversarial examples from a set of 6 “clean” images (little noise, proper lighting, clear view of the sign, etc.) and fed those into a new network trained on the same training set. The reported accuracy of the network was 41.67% (Even though our new network was trained on the same dataset that we used to generate adversarial examples, not all adversarial examples transfer successfully to new models).

Afterwards, we use our detector to extract high-level features from the image and verify if such features would indeed be present in the class determined by our network. Out of the 250 adversarial examples, our detector was able to correctly flag 75.79% of them (and we’re still improving!)

**Related Publications**


---

**Detector – Decision Tree**

- Octagon
- Diamond
- Triangle
- Color?
- Grey
- Blue
- Red
- Upside-down
- "Brown"
- Orientation?
- 2° Closest == Circle & num_vertices > 8?