Characterizing adversarial examples in deep networks with convolutional filter statistics

Fuxin Li
School of EECS
Oregon State University

web.engr.oregonstate.edu/~lif
Fooling a deep network (Szegedy et al. 2013)

- Optimizing a delta from the image to maximize a class prediction $f_{\downarrow c}(x)$

$$\max_{\Delta I} f_{\downarrow c}(I+\Delta I) - \lambda ||\Delta I||_2$$

Goldfish (95.15% confidence)
Shark (93.89% confidence)
Giant Panda (99.32% confidence)
Goldfish (95.15% confidence)

(Szegedy et al. 2013, Goodfellow et al. 2014, Nguyen et al. 2015)
Generalization of fooling

- Adversarial examples are not random
- They generalize across networks!
- Use one algorithm to generate perturbations and test on others (Luo et al. 2016)
Closer Examination of Perturbations

• Two networks used in analysis:
  • AlexNet (2012 state-of-the-art, 16% error on ImageNet challenge)
  • VGG Network (2014-2015 state-of-the-art, 7% error on ImageNet challenge)
• First part on VGG, second part on both
Generate Insights: Explore at the End

No convolution anymore
Close to final output

Use PCA (NN = linear + transformation)

Convolution network

Max pooling

Fully-connected Softmax Classification

4096 → 1000
Corruption Traces before the Fully-Connected Layer

- Do a PCA on layer-14 features (after the last convolutional layer)
Fundamental Aspect of ML

• Machine learning works only on the test data if it’s sampled from the same distribution with training data
  • No good result expected on adversarial images since never trained on it
  • Solution?
    • Enlarge training set (add adversarial examples) (Goodfellow et al. 2014)
    • Led to many GAN-type approaches
    • Or just detect the boundary of training distribution and refuse to work outside
A conservative approach

• Never do extrapolation!
  • Instead, identify intruder attempts for doing so
  • Has been studied in machine learning, e.g. self-aware learning (Li et al. 2008)
• Instead of “adding adversarial examples back to training”
  • Which never ends!
Back to Difference between Normal and Adversarial Examples

![Eigenvector Number vs. Extremal Value](image1)

![Normalized Standard Deviation](image2)
How to observe distributional statistics from a single image?

• An image is a distribution of pixels
• Each convolutional layer output is a distribution of pixels
  • K-dimensional distribution on k filter outputs
• Try not to use features to train directly (overfitting!)
• Instead, collect statistics:
  • Mean absolute value of normalized PCA coefficients
  • Minimal and maximal values
  • 25-th, 50-th and 75-th percentiles
Visualization: 2 types of adversarials

LBFGS-Adversarials (Nguyen et al. 2015)

Giant Panda

\[
\Delta f
\]

\[+0.03\]

\[
\Delta f
\]

\[+0.03\]

=  

Shark

Goldfish

EA-Adversarials (Nguyen et al. 2015)

king penguin | starfish | baseball | electric guitar

freight car | remote control | peacock | African grey
Visualization:
Single-Layer Results

- Single-layer results are OK, not fantastic
  - Imaginable with oversimplified features
  - EA-Adversarials much easier to detect

<table>
<thead>
<tr>
<th>Network Layer</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>57.5 ± 0.7</td>
<td>67.3 ± 0.7</td>
<td>70.9 ± 0.6</td>
</tr>
<tr>
<td>Network Layer</td>
<td>5th</td>
<td>6th</td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>74.9 ± 0.9</td>
<td>78.95 ± 0.6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Layer</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>72.1 ± 0.7</td>
<td>84.1 ± 0.7</td>
<td>80.3 ± 0.6</td>
</tr>
<tr>
<td>Network Layer</td>
<td>5th</td>
<td>6th</td>
<td>7th</td>
</tr>
<tr>
<td>Accuracy</td>
<td>81.4 ± 0.9</td>
<td>74.3 ± 0.6</td>
<td>73.9 ± 0.6</td>
</tr>
<tr>
<td>Network Layer</td>
<td>8th</td>
<td>9th</td>
<td>10th</td>
</tr>
<tr>
<td>Accuracy</td>
<td>74.2 ± 0.7</td>
<td>71.2 ± 0.7</td>
<td>74.3 ± 0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>93.45 ± 0.69</td>
<td>98.3 ± 0.73</td>
<td>97.9 ± 0.57</td>
</tr>
</tbody>
</table>
Classifier Cascade

- Proposed by Viola-Jones in 2001 for face detection
- Idea: discard large amount of examples that are simple to classify
- Leave those hard to classify to the next (more expensive) stage
Classifier Cascade

• 1 classifier for each convolutional layer
  • Classify on layer 1:
    • Normal: do not continue
    • Unsure: go to layer 2
    • Classify on layer 2...
Figure 7. (a) Comparison Between OpenMax detection Methods and Cascade Classifier: The blue curve represents the performance of OpenMax Method, and green curve represents the performance for Cascade Classifier. (b) Overall ROC Performance Curve of Cascade Classifier Trained on VGG-16 Network. (c) Overall ROC of data generated from EA-adversarials dataset on AlexNet.
Conclusion

- A different approach geared toward AI safety
  - Conservative
  - Avoids extrapolation
  - Try to perform “distribution tests” to test whether an example comes from input distribution
  - Classifier cascades on convolutional filter statistics work well

- Future work:
  - Generative Adversarial Network (GAN) -type approach to detect intrusion
Image Recovery for LBFGS-adversarials

• Insight: LBFGS adversarials attacks the extremal value of gradient output
• This is very specific to manipulating pixels to lower the magnitude of certain outputs
• One can counter even with simple average filtering

<table>
<thead>
<tr>
<th>Approach</th>
<th>Top-5 Accuracy (Recovered Images)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Image (Non-corrupted)</td>
<td>86.5%</td>
</tr>
<tr>
<td>3 × 3 Average Filter</td>
<td>73.0%</td>
</tr>
<tr>
<td>5 × 5 Average Filter</td>
<td>68.0%</td>
</tr>
<tr>
<td>Foveation (Object Crop MP) [16]</td>
<td>82.6%</td>
</tr>
</tbody>
</table>
Another side of the story

- It’s also not that hard to contaminate CNN!