



Adversarial Attack

By

Dr. Shiv Ram Dubey

Computer Vision and Biometrics Lab

Department of Information Technology

Indian Institute of Information Technology, Allahabad



DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.



FOOLING NEURAL NETWORKS



[Image source](#)

Generating preferred inputs

- We can use gradient ascent to generate weird-looking images to maximize activation of a given unit



dumbbell



cup

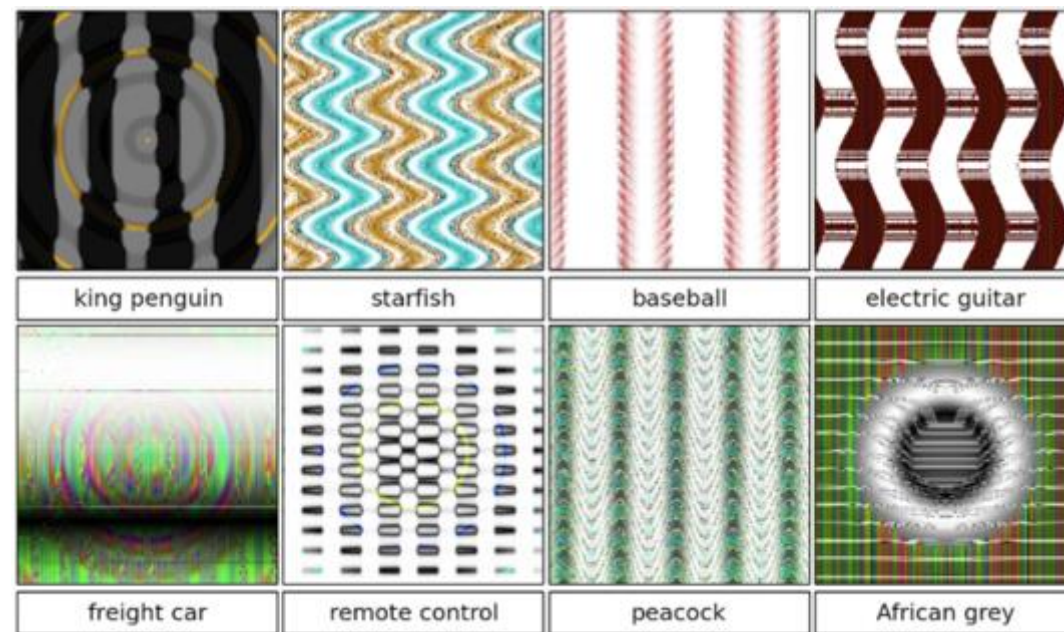
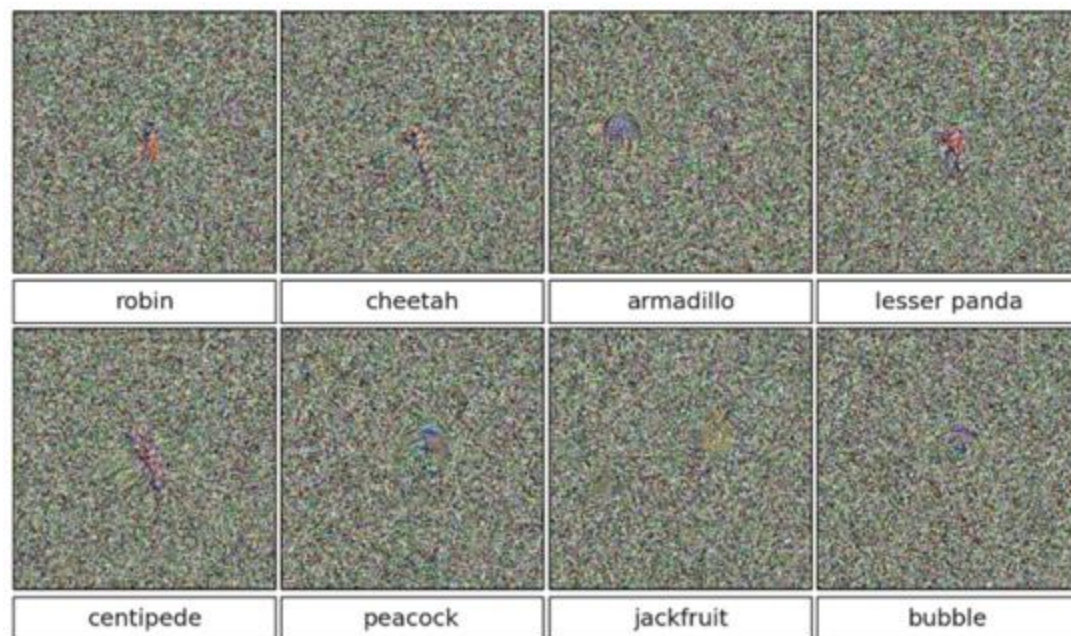


dalmatian

K. Simonyan, A. Vedaldi, and A. Zisserman, [Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps](#), ICLR 2014

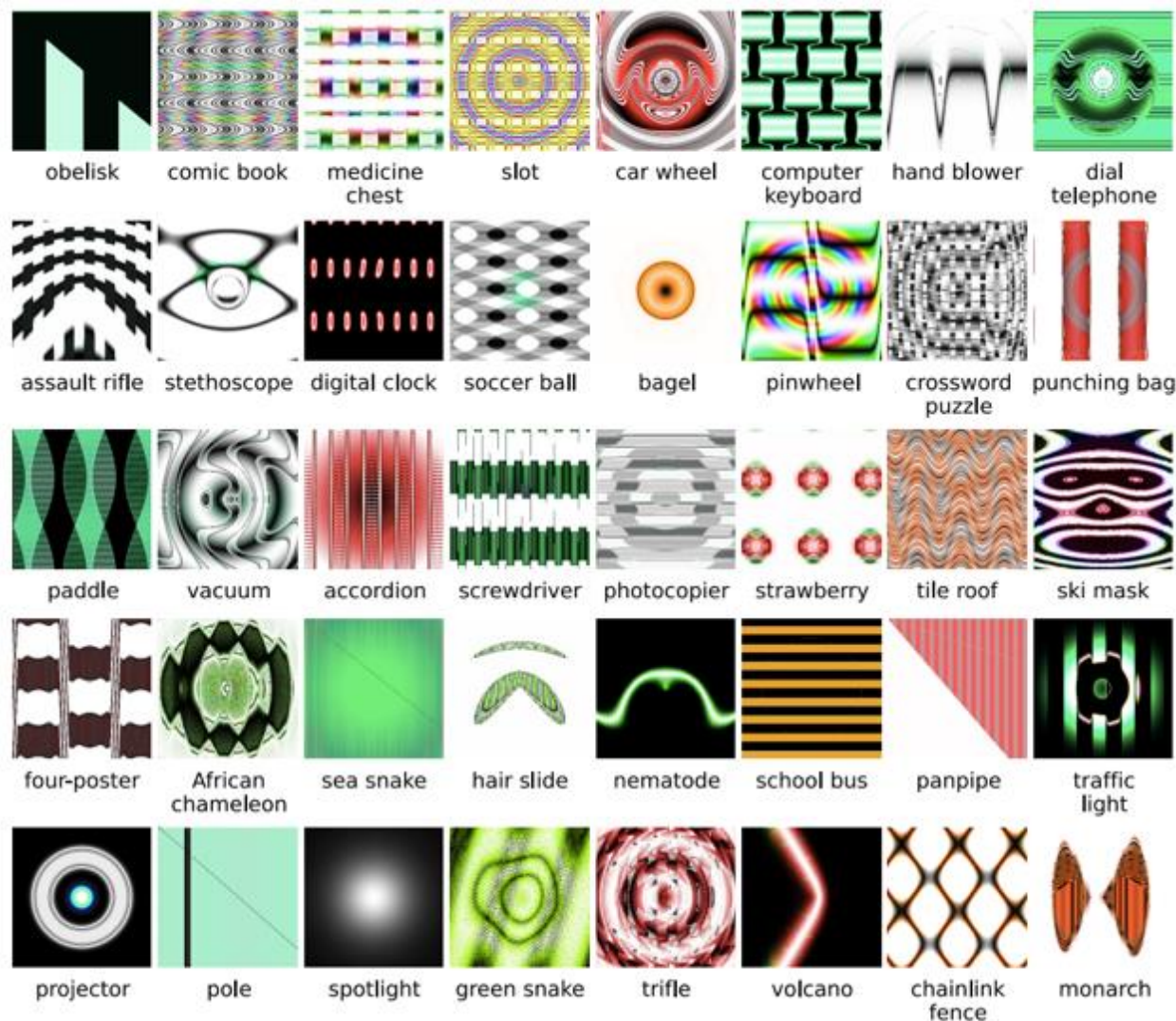
Generating preferred inputs

- Related finding: it is easy to generate meaningless images that will be classified as any given class with high confidence



A. Nguyen, J. Yosinski, J. Clune, [Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images](#), CVPR 2015

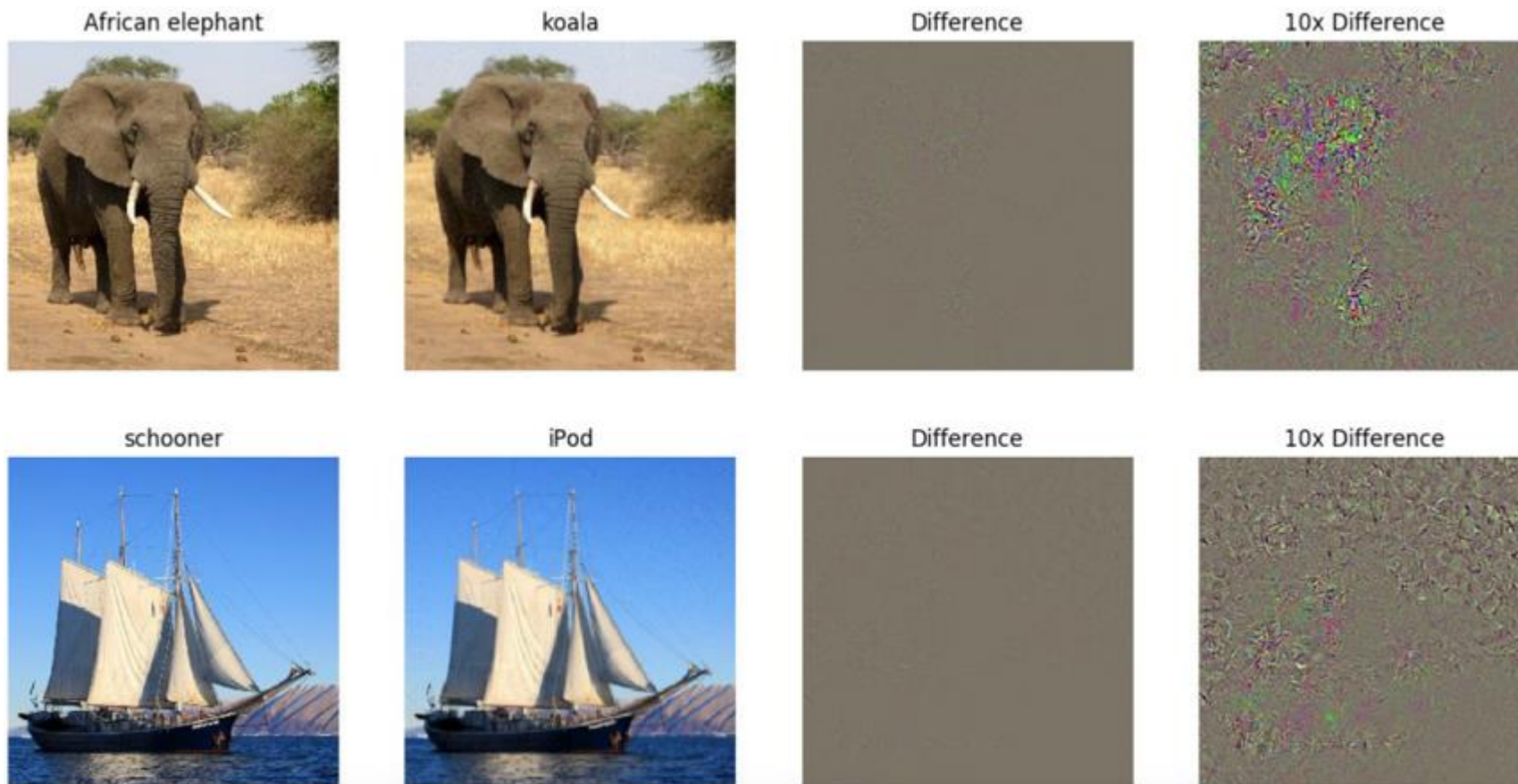
Generating preferred inputs



A. Nguyen, J. Yosinski, J. Clune, [Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images](#), CVPR 2015

Adversarial examples

- We can “fool” a neural network by imperceptibly perturbing an input image so it is misclassified



Adversarial examples: Outline

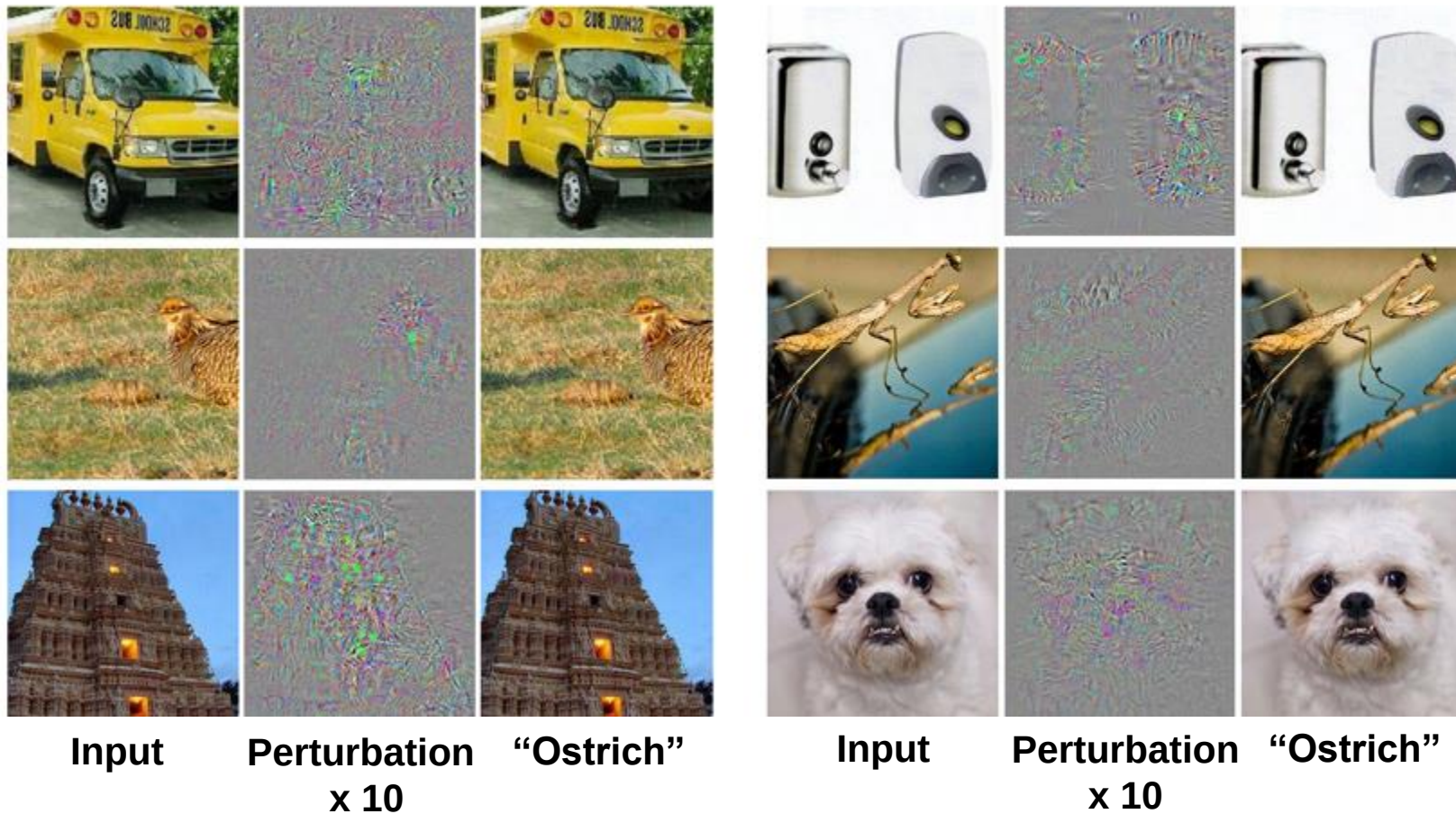
- Generating adversarial examples
 - Finding smallest “fooling” transformation
 - Gradient ascent
 - Fast gradient sign, iterative variants
 - Universal adversarial perturbations
- Why are neural networks easy to fool?
- Defending against adversarial examples
 - Adversarial training
 - Learning to reject adversarial examples
 - Robust architectures
 - Image pre-processing
- Recent Works
- “Open” topics
 - Broadening the scope of adversarial examples
 - Adversarial examples and human perception

Finding the smallest adversarial perturbation

- Start with correctly classified image x
- Find perturbation r minimizing $\|r\|_2$ such that
 - $x + r$ is misclassified (or classified as specific target class)
 - All values of $x + r$ are in the valid range
- This is constrained non-convex optimization, which the authors solve with [L-BFGS](#)
 - **Limited-memory BFGS** (**L-BFGS** or **LM-BFGS**) is an [optimization algorithm](#) in the family of [quasi-Newton methods](#)

Finding the smallest adversarial perturbation

- Sample results:



Gradient ascent

- Rather than searching for the smallest possible perturbation, it is easier to take small gradient steps in desired direction
- Decrease score (increase loss) of *correct* class y^* :

$$x \leftarrow x - \eta \frac{\partial f(x, y^*)}{\partial x} \quad \text{or} \quad x \leftarrow x + \eta \frac{\partial L(x, y^*)}{\partial x}$$

- Increase score (decrease loss) of *incorrect* target class $\hat{y}$:

$$x \leftarrow x + \eta \frac{\partial f(x, \hat{y})}{\partial x} \quad \text{or} \quad x \leftarrow x - \eta \frac{\partial L(x, \hat{y})}{\partial x}$$

Fooling a linear classifier

- Increase score of target class $\hat{y}$:

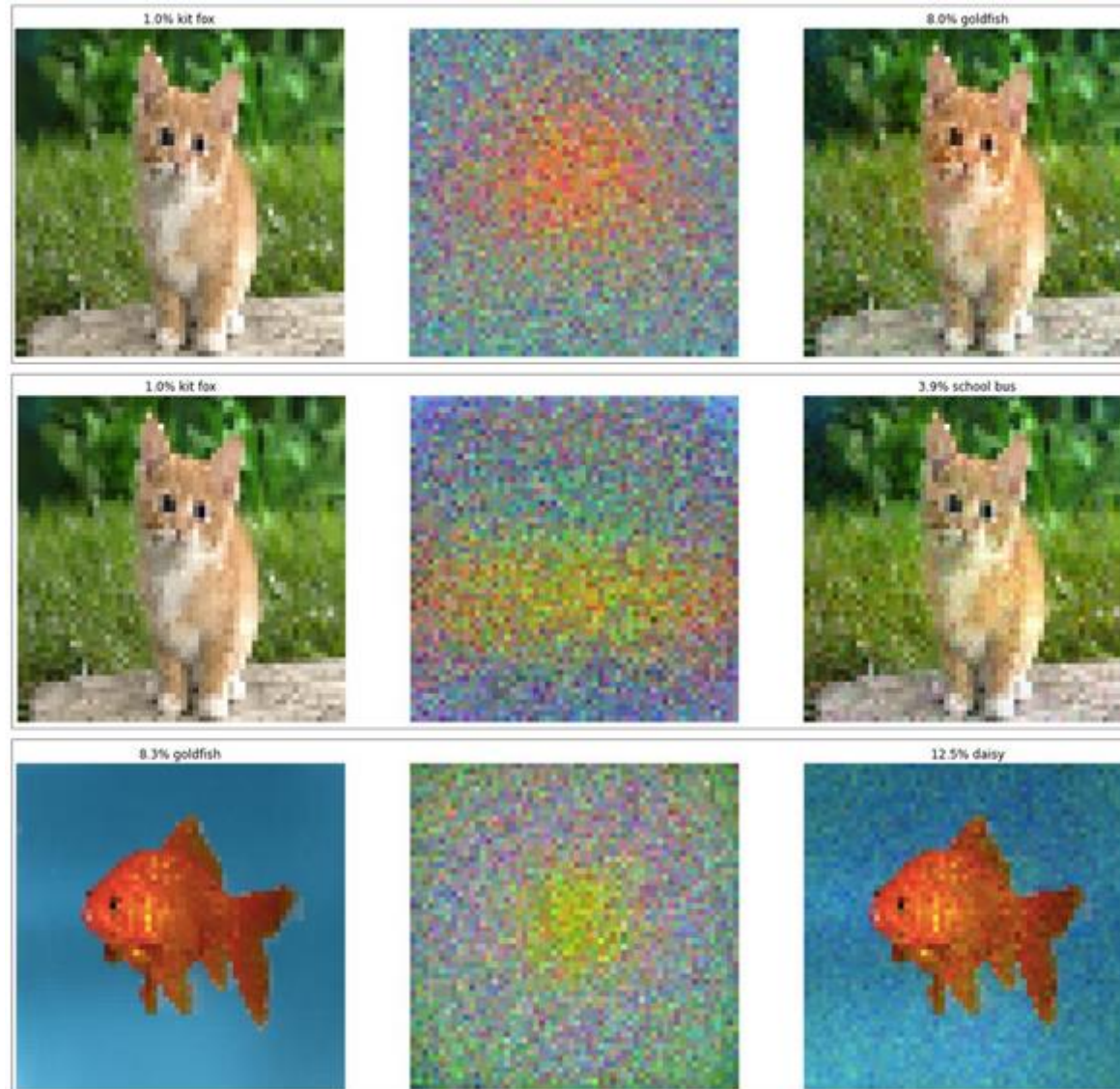
$$x \leftarrow x + \eta \frac{\partial f(x, \hat{y})}{\partial x}$$

- For a linear classifier with $f(x, \hat{y}) = w^T x$:

$$x \leftarrow x + \eta w$$

- To fool a linear classifier, add a small multiple of the target class weights to the test example

Fooling a linear classifier



Analysis of the linear case

- Response of classifier with weights w to adversarial example $x + r$:

$$w^T(x + r) = w^T x + w^T r$$

- Suppose the pixel values have precision *Epsilon* (ϵ), i.e., the classifier is normally expected to predict the same class for x and $x + r$ as long as $\|r\|_\infty \leq \epsilon$
- How to choose r to maximize the increase in activation $w^T r$ subject to $\|r\|_\infty \leq \epsilon$?

$$r = \epsilon \operatorname{sgn}(w)$$

Analysis of the linear case

- Response of classifier with weights w to adversarial example $x + r$, $r = \epsilon \operatorname{sgn}(w)$:

$$w^T(x + r) = w^T x + \epsilon w^T \operatorname{sgn}(w)$$

- If w has dimensionality d and average element magnitude m , how much will the activation increase?
 - By ϵdm , i.e., linearly as a function of d
- The higher the dimensionality, the easier it is to make many small changes to the input that cause a large change in the output

Toy example

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1

$$w^T x = -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\sigma(w^T x) = \frac{1}{1 + e^{-(-3)}} = 0.047$$

Toy example

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
$x + r$	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

$$w^T x = -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\sigma(w^T x) = \frac{1}{1 + e^{-(-3)}} = 0.047$$

$$w^T (x + r) = -3 + 10 * 0.5 = 2$$

$$\sigma(w^T (x + r)) = \frac{1}{1 + e^{-2}} = 0.88$$

Generating adversarial examples

- **Fast gradient sign method:** Find the gradient of the loss w.r.t. correct class y^* , take element-wise sign, update in resulting direction:

$$x \leftarrow x + \epsilon \operatorname{sgn} \left(\frac{\partial L(x, y^*)}{\partial x} \right)$$



x
“panda”
57.7% confidence

+ .007 ×



$\operatorname{sign}(\nabla_x J(\theta, x, y))$
“nematode”
8.2% confidence

=



$x + \epsilon \operatorname{sign}(\nabla_x J(\theta, x, y))$
“gibbon”
99.3 % confidence

Generating adversarial examples

- **Fast gradient sign method:**

$$x \leftarrow x + \epsilon \operatorname{sgn} \left(\frac{\partial L(x, y^*)}{\partial x} \right)$$

- **Iterative gradient sign method:** take multiple smaller steps until misclassified, each time clip result to be within ϵ -neighborhood of original image
- ***Least likely class method:*** try to misclassify image as class $\hat{y}$ with *smallest* initial score:

$$x \leftarrow x - \epsilon \operatorname{sgn} \left(\frac{\partial L(x, \hat{y})}{\partial x} \right)$$

Generating adversarial examples

Comparison of
methods for $\epsilon = 32$



Generating adversarial examples

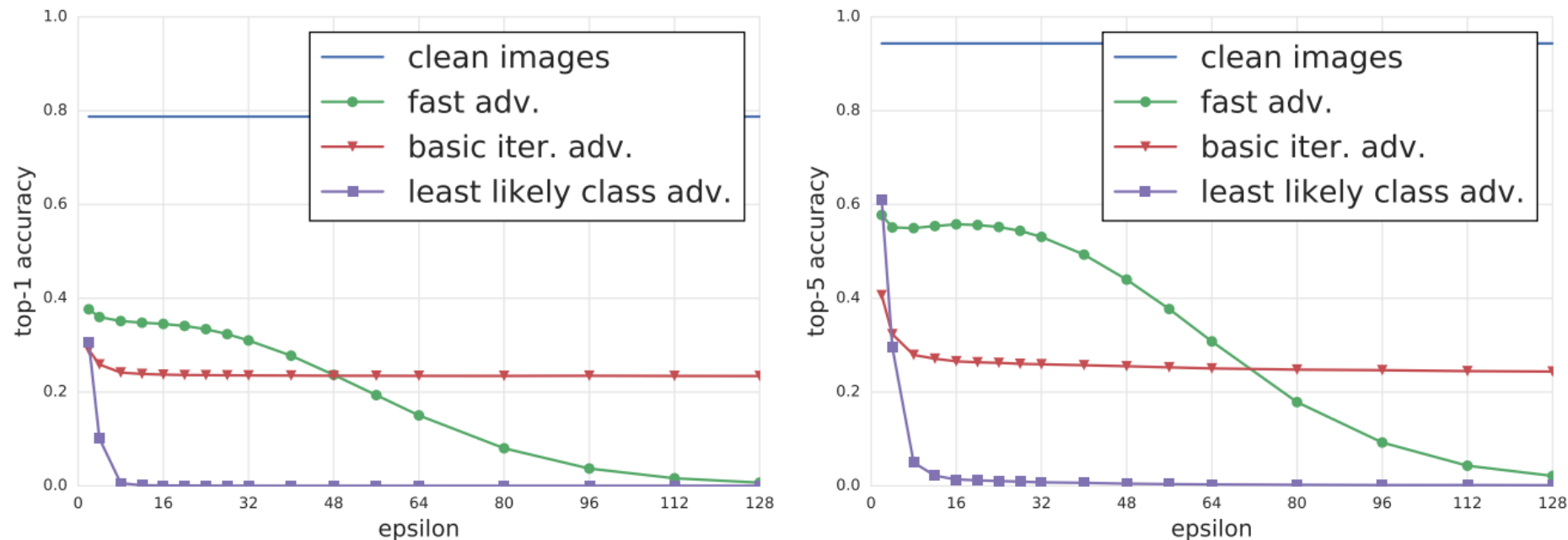


Figure 2: Top-1 and top-5 accuracy of Inception v3 under attack by different adversarial methods and different ϵ compared to “clean images” — unmodified images from the dataset. The accuracy was computed on all 50,000 validation images from the ImageNet dataset. In these experiments ϵ varies from 2 to 128.

White Box Attack vs Black Box Attack

- **White Box Attack:**
 - We **know everything about the deployed model**, e.g., inputs, model architecture, and specific model internals like weights or coefficients.
 - In most cases this specifically means that we have **access to the internal gradients** of the model.
- **Black Box Attack:**
 - We **only know the model's inputs**, and have a way to query for the output labels or confidence scores.

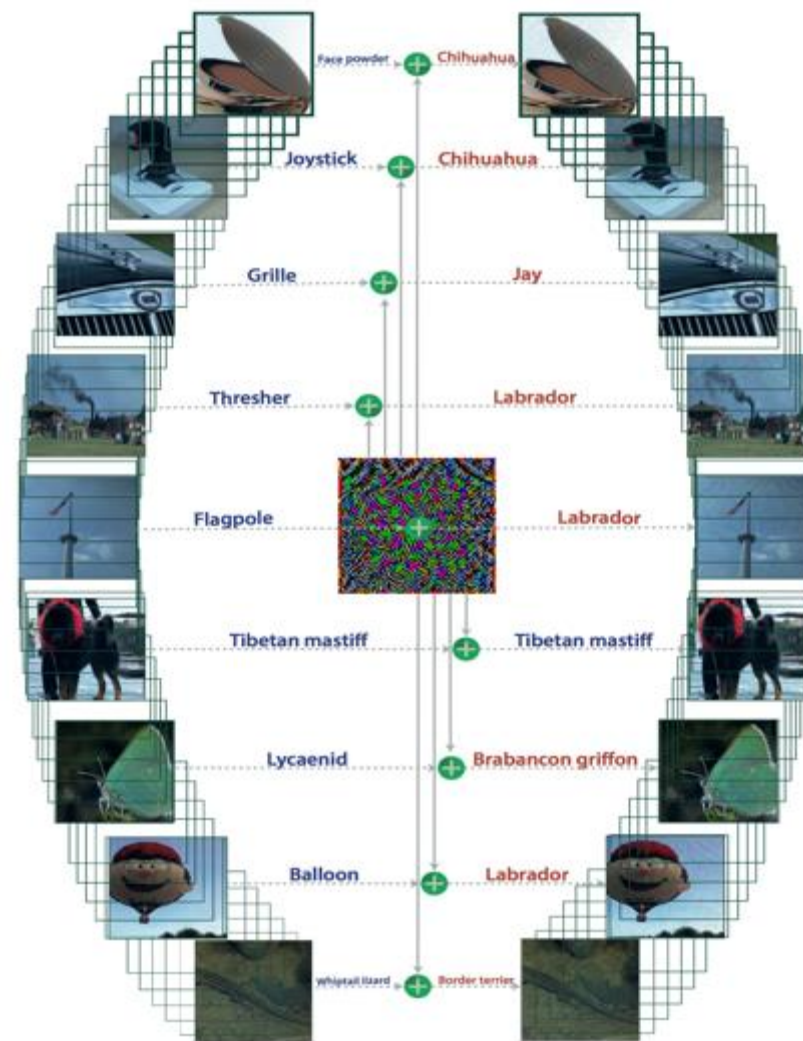
Printed adversarial examples

- “Black box” attack on a cell phone app: take a clean image, add perturbation, print out, classify with TensorFlow Camera Demo app



Universal adversarial perturbations

- Goal: for a given network, find an *image-independent* perturbation vector that causes *all* images to be misclassified with high probability



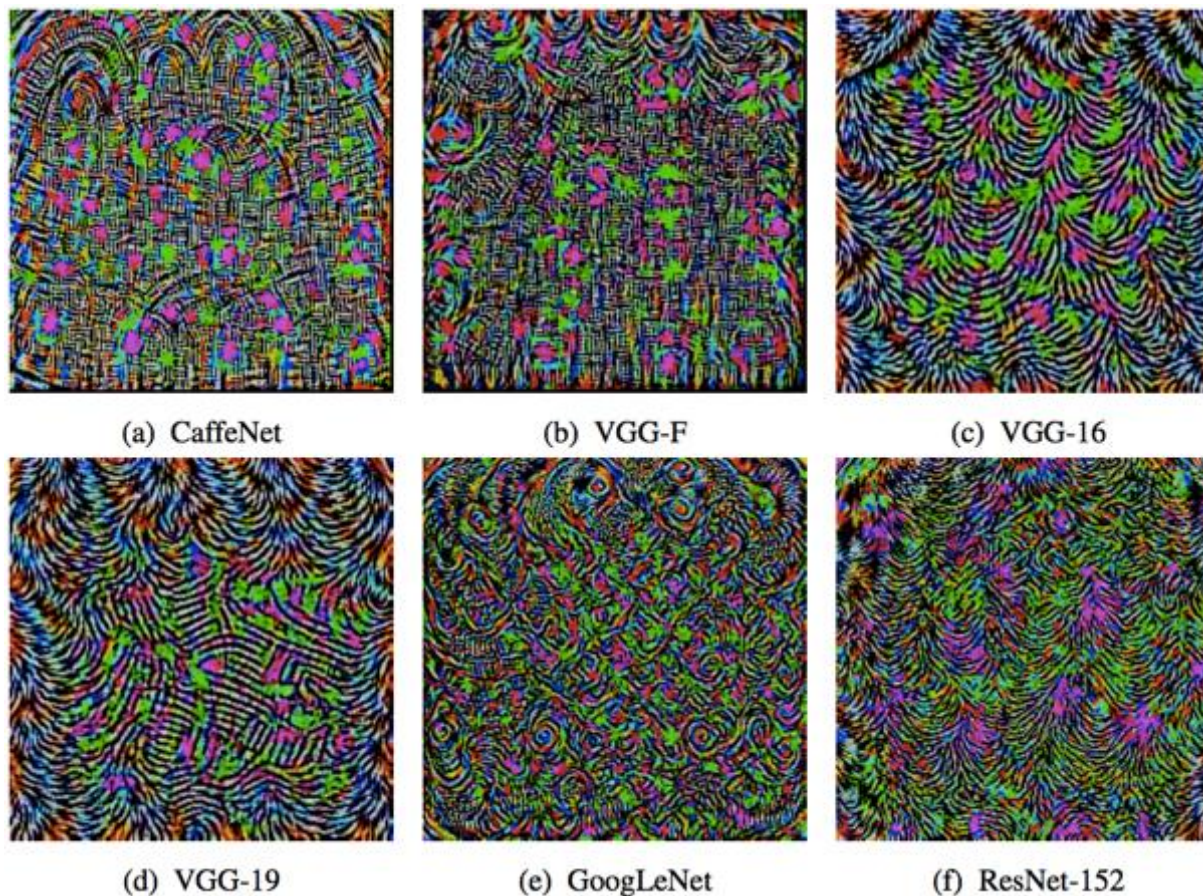
Universal adversarial perturbations

Approach:

- Start with $r = 0$
- Cycle through training examples x_i (in multiple passes)
 - If $x_i + r$ is misclassified, skip to x_{i+1}
 - Find minimum perturbation Δr that takes $x_i + r + \Delta r$ to another class
 - Update $r \leftarrow r + \Delta r$, enforce $\|r\| \leq \epsilon$
- Terminate when fooling rate on training examples reaches target value

Universal adversarial perturbations

- Perturbation vectors computed from different architectures:



Universal adversarial perturbations

- Universal perturbations turn out to generalize well across models!

Fooling rate when computing a perturbation for one model (rows)
and testing it on others (columns)

	VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19	ResNet-152
VGG-F	93.7%	71.8%	48.4%	42.1%	42.1%	47.4 %
CaffeNet	74.0%	93.3%	47.7%	39.9%	39.9%	48.0%
GoogLeNet	46.2%	43.8%	78.9%	39.2%	39.8%	45.5%
VGG-16	63.4%	55.8%	56.5%	78.3%	73.1%	63.4%
VGG-19	64.0%	57.2%	53.6%	73.5%	77.8%	58.0%
ResNet-152	46.3%	46.3%	50.5%	47.0%	45.5%	84.0%

Black-box adversarial examples

- Suppose the adversary can only query a target network with chosen inputs and observe the outputs
- Key idea: learn substitute for target network using synthetic input data, use substitute network to craft adversarial examples
- Successfully attacked third-party APIs from MetaMind, Amazon, and Google, but only on low-res digit and street sign images

Properties of adversarial examples

- For any input image, it is usually easy to generate a very similar image that gets misclassified by the same network
- To obtain an adversarial example, one does not need to do precise gradient ascent
- Adversarial images can (somewhat) survive transformations like being printed and photographed
- It is possible to attack many images with the same perturbation
- Adversarial examples that can fool one network have a high chance of fooling a network with different parameters and even architecture

Why are deep networks easy to fool?

- Networks are “too linear”: it is easy to manipulate output in a predictable way given the input
- The input dimensionality is high, so one can get a large change in the output by changing individual inputs by small amounts
- Neural networks can fit anything, but nothing prevents them from behaving erratically between training samples
 - Counter-intuitively, a network can both generalize well on natural images and be susceptible to adversarial examples
- Adversarial examples generalize well because different models learn similar functions when trained to perform the same task (or because adversarial examples are a function of the data rather than of the network)?

Adversarial examples: Outline

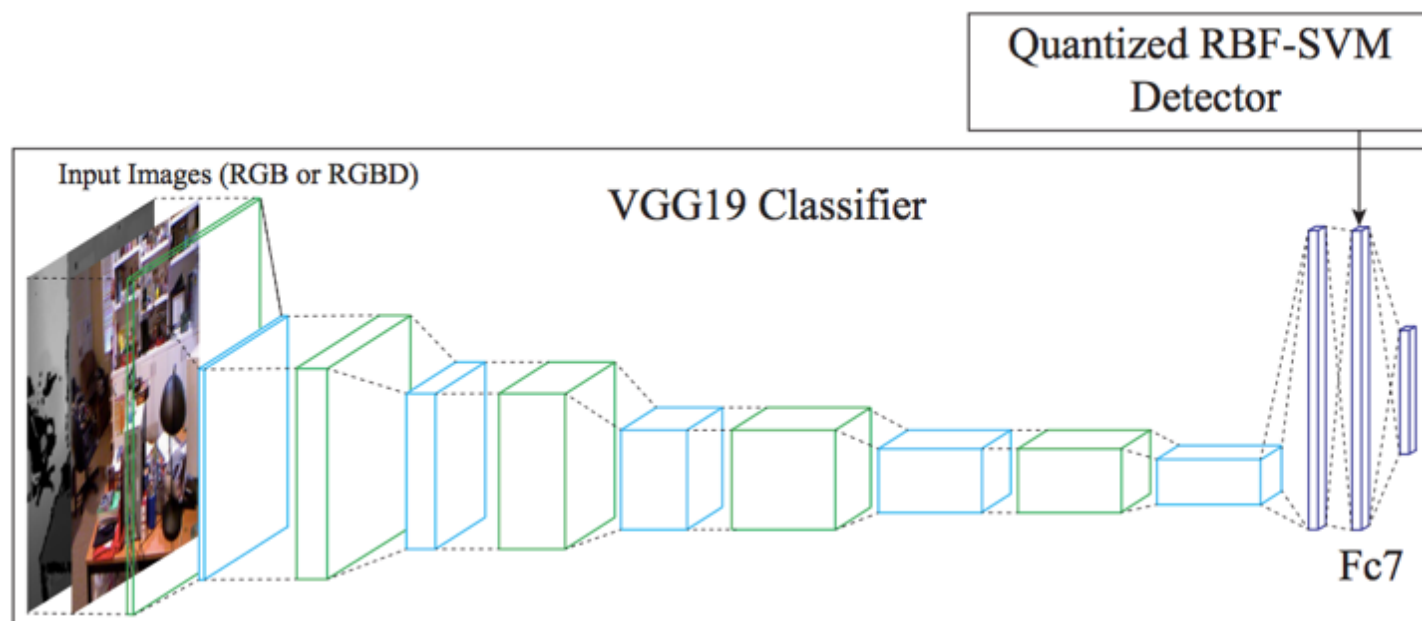
- Generating adversarial examples
 - Finding smallest “fooling” transformation
 - Gradient ascent
 - Fast gradient sign, iterative variants
 - Universal adversarial perturbations
- Why are neural networks easy to fool?
- Defending against adversarial examples
 - Adversarial training
 - Learning to reject adversarial examples
 - Robust architectures
 - Image pre-processing

Defending against adversarial examples

- Adversarial training: networks can be made somewhat resistant by augmenting or regularizing training with adversarial examples ([Goodfellow et al. 2015](#), [Tramer et al. 2018](#))

Defending against adversarial examples

- Train a separate model to reject adversarial examples:
SafetyNet
 - uses radial basis function kernel (RBF) - SVM



Defending against adversarial examples

- Robust architectures

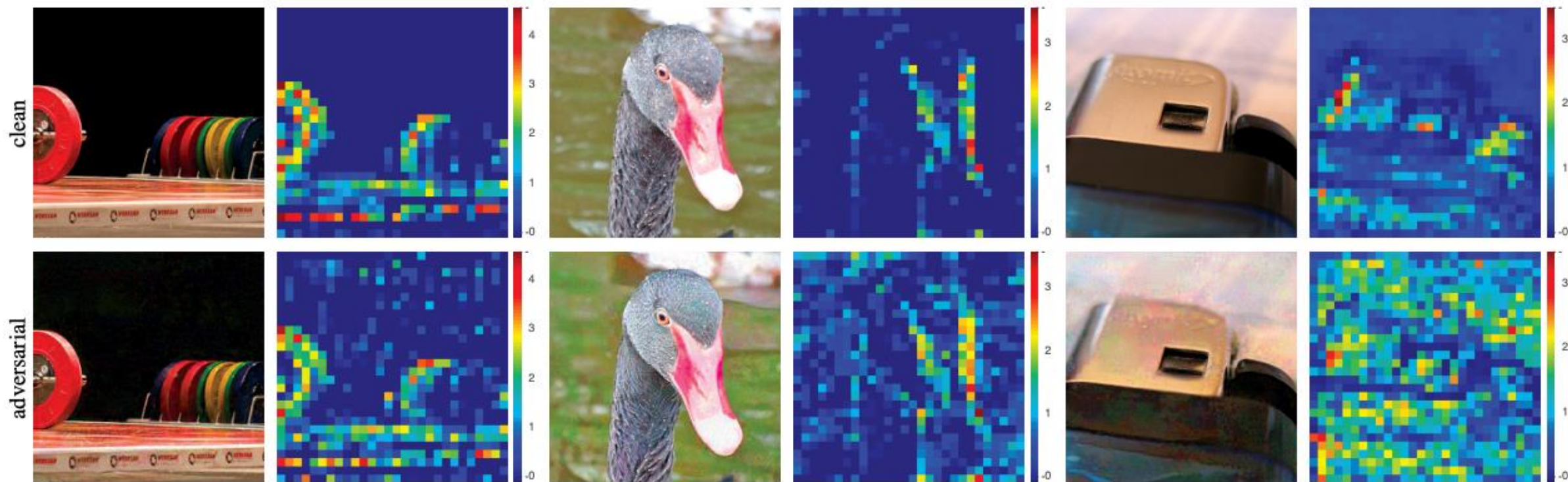
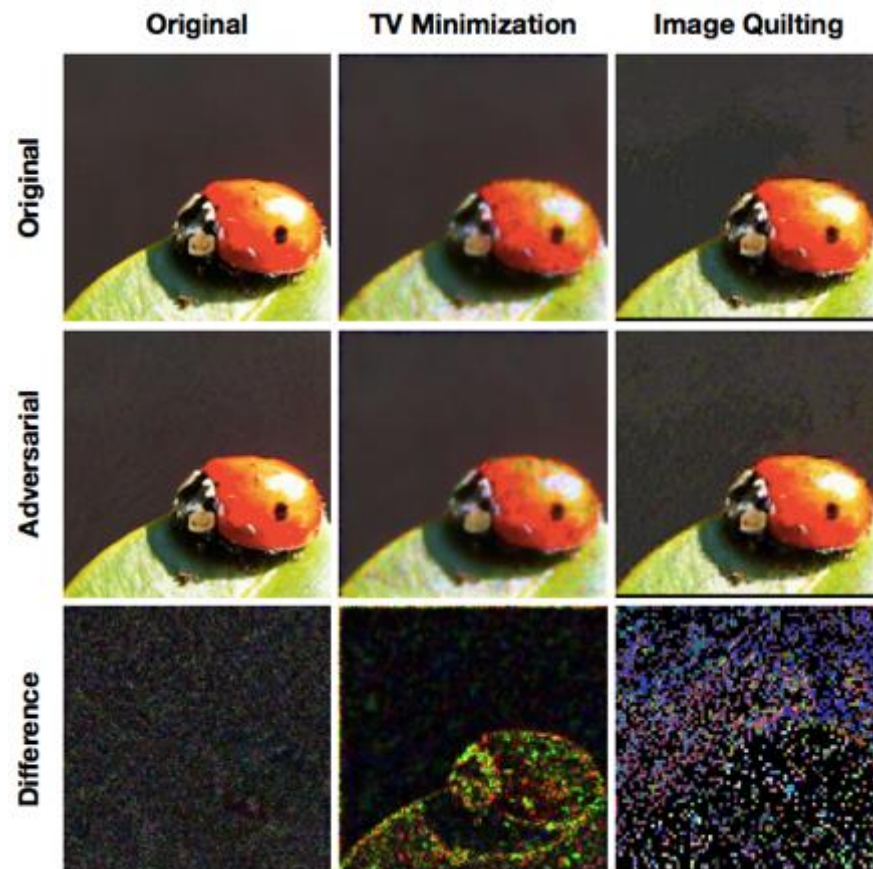


Figure 2. More examples similar to Figure 1. We show feature maps corresponding to clean images (top) and to their adversarial perturbed versions (bottom). The feature maps for each pair of examples are from the same channel of a res_3 block in the same ResNet-50 trained on clean images. The attacker has a maximum perturbation $\epsilon = 16$ in the pixel domain.

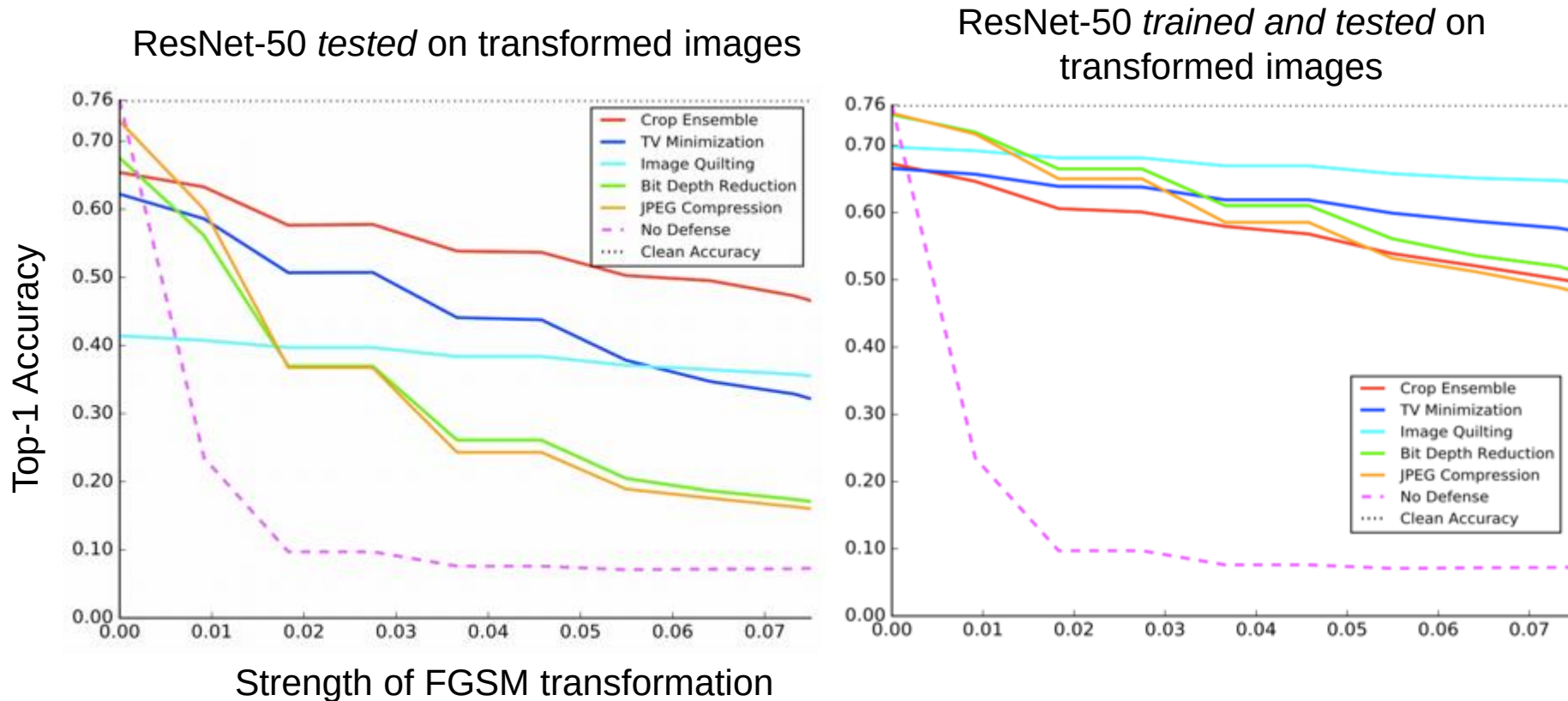
Defending against adversarial examples

- Pre-process input images to disrupt adversarial perturbations



Defending against adversarial examples

- Pre-process input images to disrupt adversarial perturbations



Adversarial examples: Outline

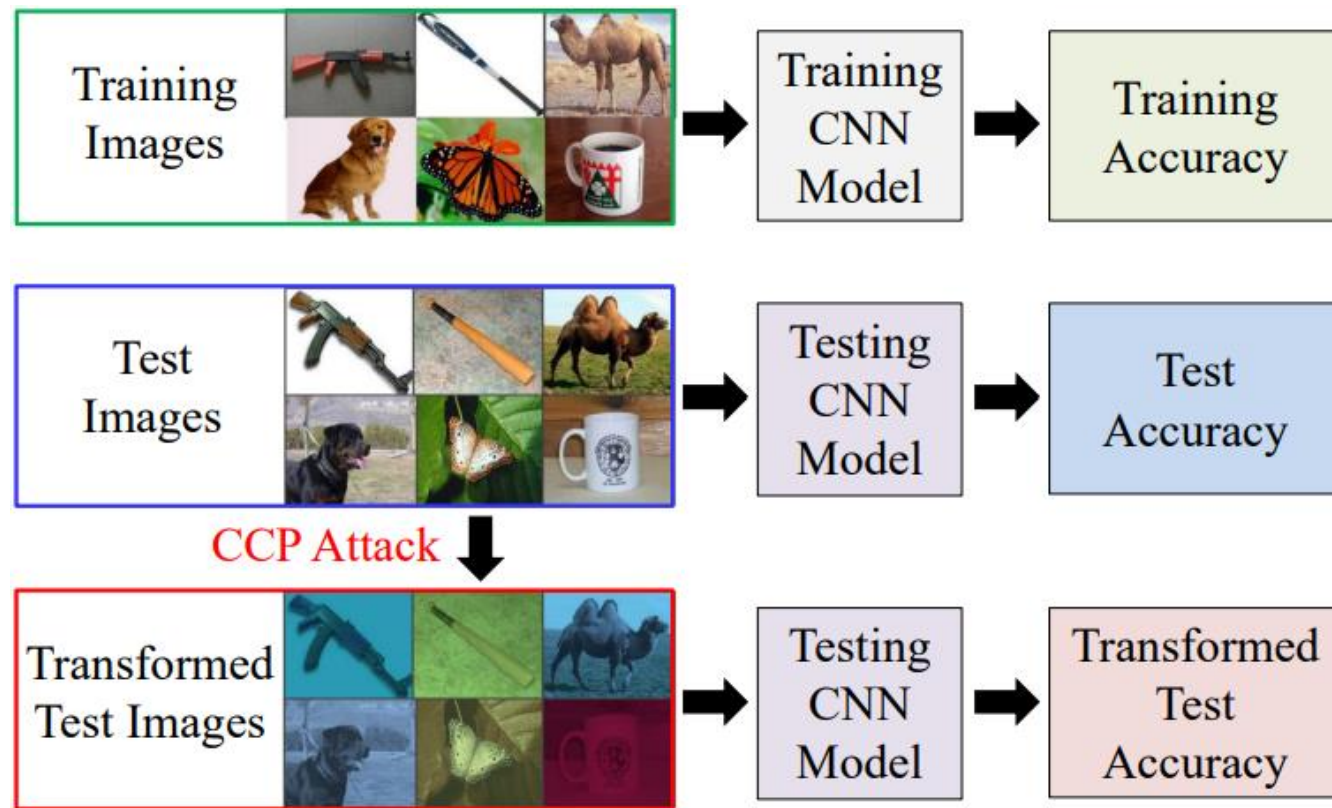
- Generating adversarial examples
 - Finding smallest “fooling” transformation
 - Gradient ascent
 - Fast gradient sign, iterative variants
 - Universal adversarial perturbations
- Why are neural networks easy to fool?
- Defending against adversarial examples
 - Adversarial training
 - Learning to reject adversarial examples
 - Robust architectures
 - Image pre-processing
- Recent Works

Color Channel Perturbation Attack

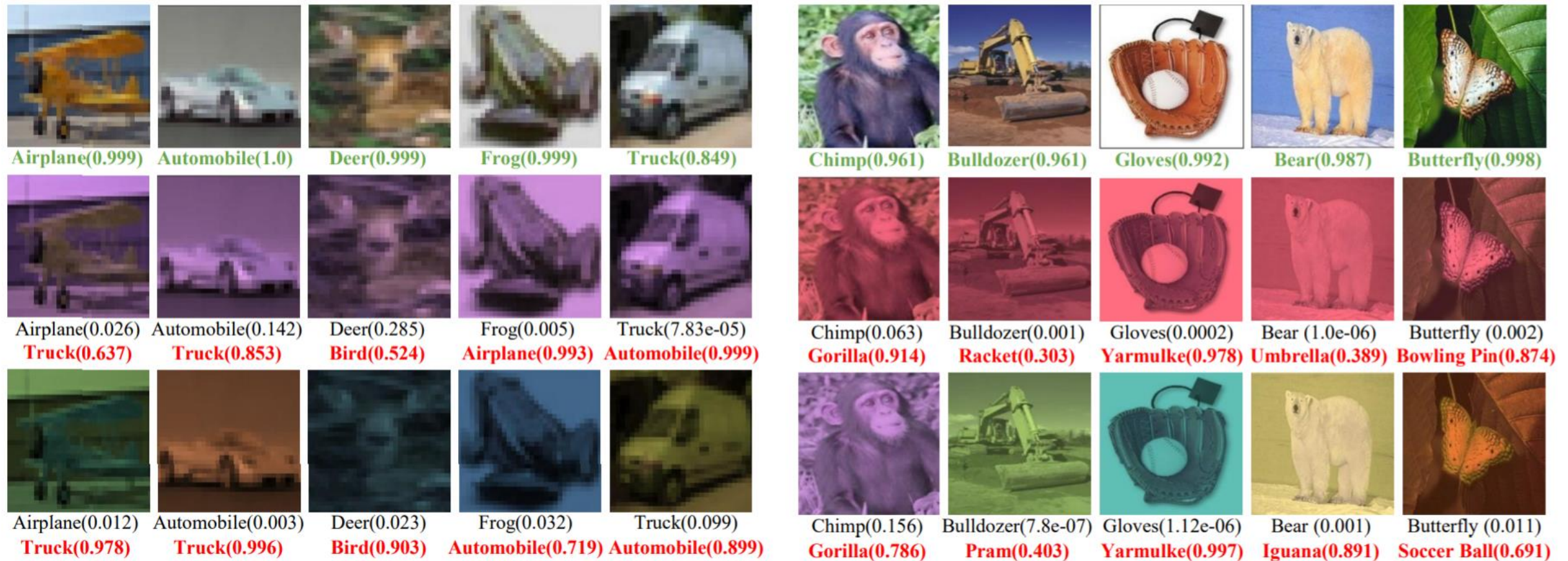
$$R_i^T = s \times \left(\frac{\alpha_i^r \times R_i + \alpha_i^g \times G_i + \alpha_i^b \times B_i}{3} \right) + b$$

$$G_i^T = s \times \left(\frac{\beta_i^r \times R_i + \beta_i^g \times G_i + \beta_i^b \times B_i}{3} \right) + b$$

$$B_i^T = s \times \left(\frac{\gamma_i^r \times R_i + \gamma_i^g \times G_i + \gamma_i^b \times B_i}{3} \right) + b$$



Color Channel Perturbation Attack



1st Row: Clean Image, 2nd Row: CCP-Fixed Attack, 3rd Row: CCP-Variable Attack

K. Jayendra, S.R. Dubey, S. Chakraborty, [Color Channel Perturbation Attacks for Fooling Convolutional Neural Networks and A Defense Against Such Attacks](#), IEEE TAI 2020

Color Channel Perturbation Attack

Accuracy (%) over original CIFAR10 training data and different test sets

30 Trials	VGG16		ResNet56		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	76.49	75.92	78.38	77.58	60.23	56.46
STD	9.16	0.29	7.54	0.29	10.93	0.35
Minimum	53.59	75.20	59.99	76.94	37.77	55.68
Maximum	88.31	76.36	88.55	78.34	77.95	56.99
Original	93.58		91.44		92.72	

Accuracy (%) over original Caltech256 training data and different test sets

30 Trials	VGG19		ResNet18		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	28.76	28.01	38.01	41.58	38.10	38.56
STD	6.65	0.36	9.60	0.43	5.98	2.25
Minimum	16.14	27.29	14.77	40.89	26.73	32.67
Maximum	40.55	28.80	51.70	42.61	49.32	42.51
Original	50.42		66.73		64.83	

Accuracy (%) over original TinyImageNet training data and different test sets

30 Trials	VGG16		ResNet101		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	13.31	14.07	26.09	26.72	30.55	31.09
STD	4.04	0.29	4.81	0.29	4.09	0.25
Minimum	3.34	13.48	12.95	26.21	18.26	27.90
Maximum	20.26	14.70	33.54	27.38	37.86	29.23
Original	42.55		59.19		62.52	

Color Channel Perturbation Attack

Accuracy (%) over CCP augmented CIFAR10 training data and different test sets

30 Trials	VGG16		ResNet56		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	90.81	90.89	89.00	88.92	88.83	88.67
STD	0.59	0.61	0.35	0.67	0.35	0.12
Minimum	89.16	88.09	88.31	86.63	88.23	88.39
Maximum	91.35	91.43	89.67	89.59	89.48	88.95
Original	91.42		90.15		90.61	

Accuracy (%) over CCP augmented Caltech256 training data and different test sets

30 Trials	VGG19		ResNet18		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	40.14	40.07	63.28	63.36	60.93	60.64
STD	3.51	0.25	0.72	0.24	1.54	0.16
Minimum	31.00	39.63	60.72	62.73	57.28	60.35
Maximum	45.18	40.68	64.16	63.69	62.78	61.11
Original	52.57		64.87		62.58	

Accuracy (%) over CCP augmented TinyImageNet training data and different test sets

30 Trials	VGG16		ResNet101		DenseNet121	
	CCP_f	CCP_v	CCP_f	CCP_v	CCP_f	CCP_v
Mean	34.02	33.82	51.87	51.84	55.64	55.49
STD	1.09	0.23	1.05	0.25	1.02	0.23
Minimum	31.90	33.43	49.19	51.38	53.33	54.98
Maximum	35.99	34.47	53.37	52.35	57.21	55.92
Original	43.73		59.65		63.41	

Color Channel Perturbation Attack

with original training data

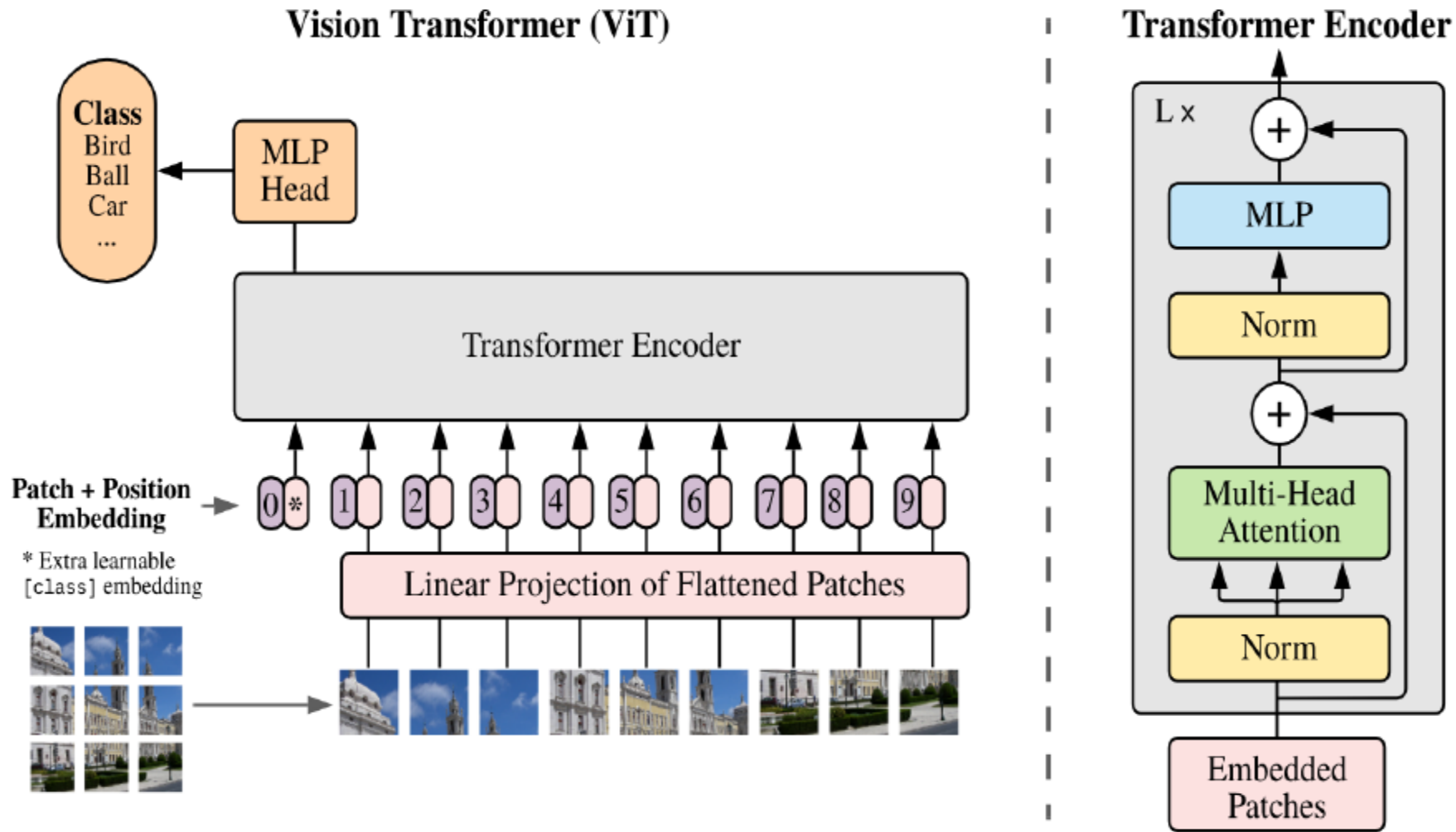
Accuracy over CIFAR10 Dataset			
Type of Attack	VGG16	ResNet56	DenseNet121
Without Attack	93.58 $\pm$ 0.00	91.44 $\pm$ 0.00	92.72 $\pm$ 0.00
One Pixel Attack	92.81 $\pm$ 0.10 ($\downarrow$ 0.82%)	89.40 $\pm$ 0.16 ($\downarrow$ 2.23%)	92.21 $\pm$ 0.07 ($\downarrow$ 0.55%)
Adversarial Attack	82.61 $\pm$ 0.21 ($\downarrow$ 11.72%)	79.78 $\pm$ 0.79 ($\downarrow$ 12.75%)	67.43 $\pm$ 0.21 ($\downarrow$ 27.28%)
CCP Attack (CCP_f)	76.49 $\pm$ 9.16 ($\downarrow$ 18.26%)	78.38 $\pm$ 7.54 ($\downarrow$ 14.28%)	60.23 $\pm$ 10.9 ($\downarrow$ 35.04%)
CCP Attack (CCP_v)	75.92 $\pm$ 0.29 ($\downarrow$ 18.87%)	77.58 $\pm$ 0.29 ($\downarrow$ 15.16%)	56.46 $\pm$ 0.35 ($\downarrow$ 39.11%)
Accuracy over Caltech256 Dataset			
Type of Attack	VGG19	ResNet18	DenseNet121
Without Attack	50.42 $\pm$ 0.00	66.73 $\pm$ 0.00	64.83 $\pm$ 0.00
One Pixel Attack	50.43 $\pm$ 0.02 ($\uparrow$ 0.02%)	66.64 $\pm$ 0.05 ($\downarrow$ 0.13%)	64.79 $\pm$ 0.03 ($\downarrow$ 0.06%)
Adversarial Attack	48.11 $\pm$ 0.11 ($\downarrow$ 4.58%)	64.69 $\pm$ 0.13 ($\downarrow$ 3.06%)	63.58 $\pm$ 0.13 ($\downarrow$ 1.93%)
CCP Attack (CCP_f)	28.76 $\pm$ 6.65 ($\downarrow$ 42.96%)	38.01 $\pm$ 9.60 ($\downarrow$ 43.04%)	38.10 $\pm$ 5.98 ($\downarrow$ 41.23%)
CCP Attack (CCP_v)	28.01 $\pm$ 0.36 ($\downarrow$ 44.45%)	41.58 $\pm$ 0.43 ($\downarrow$ 37.69%)	38.56 $\pm$ 2.25 ($\downarrow$ 40.52%)
Accuracy over TinyImageNet Dataset			
Type of Attack	VGG16	ResNet101	DenseNet121
Without Attack	42.55 $\pm$ 0.00	59.19 $\pm$ 0.00	62.52 $\pm$ 0.00
One Pixel Attack	42.61 $\pm$ 0.09 ($\uparrow$ 0.14%)	59.01 $\pm$ 0.08 ($\downarrow$ 0.30%)	62.44 $\pm$ 0.08 ($\downarrow$ 1.28%)
Adversarial Attack	40.26 $\pm$ 0.19 ($\downarrow$ 5.38%)	56.44 $\pm$ 0.14 ($\downarrow$ 4.65%)	60.01 $\pm$ 0.16 ($\downarrow$ 4.01%)
CCP Attack (CCP_f)	13.31 $\pm$ 4.04 ($\downarrow$ 68.72%)	26.09 $\pm$ 4.81 ($\downarrow$ 55.92%)	30.55 $\pm$ 4.09 ($\downarrow$ 51.14%)
CCP Attack (CCP_v)	14.07 $\pm$ 0.29 ($\downarrow$ 66.93%)	26.72 $\pm$ 0.29 ($\downarrow$ 54.86%)	31.09 $\pm$ 0.25 ($\downarrow$ 50.27%)

Color Channel Perturbation Attack

with CCP transformation in the training data

Accuracy over CIFAR10 Dataset			
Type of Attack	VGG16	ResNet56	DenseNet121
Without Attack	91.42 $\pm$ 0.00 ($\downarrow$ 2.13%)	90.15 $\pm$ 0.00 ($\downarrow$ 1.41%)	90.61 $\pm$ 0.00 ($\downarrow$ 2.28%)
One Pixel Attack	90.57 $\pm$ 0.08 ($\downarrow$ 2.41%)	89.07 $\pm$ 0.17 ($\downarrow$ 0.37%)	89.12 $\pm$ 0.12 ($\downarrow$ 3.35%)
Adversarial Attack	81.36 $\pm$ 0.18 ($\downarrow$ 1.51%)	80.62 $\pm$ 0.61 ($\uparrow$ 1.05%)	75.08 $\pm$ 0.21 ($\uparrow$ 11.35%)
CCP Attack (CCP_f)	90.81 $\pm$ 0.59 ($\uparrow$ 18.72%)	89.00 $\pm$ 0.35 ($\uparrow$ 13.55%)	88.83 $\pm$ 0.35 ($\uparrow$ 47.48%)
CCP Attack (CCP_v)	90.89 $\pm$ 0.61 ($\uparrow$ 19.72%)	88.92 $\pm$ 0.67 ($\uparrow$ 14.62%)	88.67 $\pm$ 0.12 ($\uparrow$ 57.05%)
Accuracy over Caltech256 Dataset			
Type of Attack	VGG19	ResNet18	DenseNet121
Without Attack	52.57 $\pm$ 0.00 ($\uparrow$ 4.26%)	64.87 $\pm$ 0.00 ($\downarrow$ 2.79%)	63.41 $\pm$ 0.00 ($\downarrow$ 2.19%)
One Pixel Attack	52.33 $\pm$ 0.03 ($\uparrow$ 3.77%)	64.82 $\pm$ 0.05 ($\downarrow$ 2.73%)	62.49 $\pm$ 0.04 ($\downarrow$ 3.55%)
Adversarial Attack	49.55 $\pm$ 0.11 ($\uparrow$ 2.99%)	63.04 $\pm$ 0.15 ($\downarrow$ 2.55%)	60.36 $\pm$ 0.14 ($\downarrow$ 5.06%)
CCP Attack (CCP_f)	40.14 $\pm$ 3.51 ($\uparrow$ 39.57%)	63.28 $\pm$ 0.72 ($\uparrow$ 66.48%)	60.93 $\pm$ 1.54 ($\uparrow$ 59.92%)
CCP Attack (CCP_v)	40.07 $\pm$ 0.25 ($\uparrow$ 43.06%)	63.36 $\pm$ 0.24 ($\uparrow$ 52.38%)	60.64 $\pm$ 0.16 ($\uparrow$ 57.26%)
Accuracy over TinyImageNet Dataset			
Type of Attack	VGG16	ResNet101	DenseNet121
Without Attack	43.73 $\pm$ 0.00 ($\uparrow$ 2.77%)	59.65 $\pm$ 0.00 ($\uparrow$ 0.78%)	63.41 $\pm$ 0.00 ($\uparrow$ 1.42%)
One Pixel Attack	43.55 $\pm$ 0.09 ($\uparrow$ 2.21%)	59.59 $\pm$ 0.09 ($\uparrow$ 0.98%)	63.14 $\pm$ 0.11 ($\uparrow$ 1.12%)
Adversarial Attack	40.87 $\pm$ 0.14 ($\uparrow$ 1.52%)	56.88 $\pm$ 0.19 ($\uparrow$ 0.78%)	60.53 $\pm$ 0.18 ($\uparrow$ 0.87%)
CCP Attack (CCP_f)	34.02 $\pm$ 1.09 ($\uparrow$ 155.60%)	51.87 $\pm$ 1.05 ($\uparrow$ 98.81%)	55.64 $\pm$ 1.02 ($\uparrow$ 82.13%)
CCP Attack (CCP_v)	33.82 $\pm$ 0.23 ($\uparrow$ 140.37%)	51.84 $\pm$ 0.25 ($\uparrow$ 94.01%)	55.49 $\pm$ 0.23 ($\uparrow$ 78.48%)

Color Channel Perturbation Attack on ViT



Color Channel Perturbation Attack on ViT

ViT –
Vision Transformer

PiT –
Pooling-based
Vision Transformer

CaiT –
Class-Attention in
Image Transformers

Accuracy over CIFAR10 Dataset			
Model	Without Attack	CCP Attack (CCP_f)	CCP Attack (CCP_v)
VGG16	93.58	76.49 $\pm$ 9.16 ($\downarrow$ 18.26%)	75.92 $\pm$ 0.29 ($\downarrow$ 18.87%)
ResNet256	91.44	78.38 $\pm$ 7.54 ($\downarrow$ 14.28%)	77.58 $\pm$ 0.29 ($\downarrow$ 15.16%)
DenseNet121	92.72	60.23 $\pm$ 10.9 ($\downarrow$ 35.04%)	56.46 $\pm$ 0.35 ($\downarrow$ 39.11%)
ViT	94.4	89.15 $\pm$ 22.47 ($\downarrow$ 5.50%)	89.70 $\pm$ 0.05 ($\downarrow$ 4.97%)
PiT	94.46	89.41 $\pm$ 19.18 ($\downarrow$ 5.34%)	89.87 $\pm$ 0.07 ($\downarrow$ 4.85%)
CaiT	96.14	89.70 $\pm$ 17.56 ($\downarrow$ 06.69%)	90.58 $\pm$ 0.06 ($\downarrow$ 05.78%)
Accuracy over Caltech256 Dataset			
VGG16	50.42	28.76 $\pm$ 6.65 ($\downarrow$ 42.96%)	28.01 $\pm$ 0.36 ($\downarrow$ 44.45%)
ResNet256	66.73	38.01 $\pm$ 9.60 ($\downarrow$ 43.04%)	41.58 $\pm$ 0.43 ($\downarrow$ 37.69%)
DenseNet121	64.83	38.10 $\pm$ 5.98 ($\downarrow$ 41.23%)	38.56 $\pm$ 2.25 ($\downarrow$ 40.52%)
ViT	60.2	25.94 $\pm$ 1.14 ($\downarrow$ 56.91%)	28.47 $\pm$ 0.05 ($\downarrow$ 52.70%)
PiT	69.39	23.69 $\pm$ 7.54 ($\downarrow$ 65.85%)	25.35 $\pm$ 8.43 ($\downarrow$ 63.46%)
CaiT	63.15	26.86 $\pm$ 4.69 ($\downarrow$ 57.46%)	28.14 $\pm$ 1.08 ($\downarrow$ 55.43%)
Accuracy over TinyImageNet Dataset			
VGG16	42.55	13.31 $\pm$ 4.04 ($\downarrow$ 68.72%)	14.07 $\pm$ 0.29 ($\downarrow$ 66.93%)
ResNet256	59.19	26.09 $\pm$ 4.81 ($\downarrow$ 55.92%)	26.72 $\pm$ 0.29 ($\downarrow$ 54.86%)
DenseNet121	62.52	30.55 $\pm$ 4.09 ($\downarrow$ 51.14%)	31.09 $\pm$ 0.25 ($\downarrow$ 50.27%)
ViT	62.39	19.87 $\pm$ 9.86 ($\downarrow$ 68.15%)	22.61 $\pm$ 7.77 ($\downarrow$ 63.70%)
PiT	75.32	26.97 $\pm$ 2.33 ($\downarrow$ 64.19%)	27.70 $\pm$ 3.53 ($\downarrow$ 63.22%)
CaiT	60.23	25.46 $\pm$ 2.80 ($\downarrow$ 57.72%)	25.92 $\pm$ 3.29 ($\downarrow$ 56.96%)

Non-Uniform Illumination Attack

Mask ID	Mask	Region of perturbation
Mask 1	$a = ((u - x) \times \frac{30}{u}) + ((v - y) \times \frac{30}{v}) + ((u - y) \times \frac{20}{u}) + ((v - y) \times \frac{20}{v})$	Focused more on the left side
Mask 2	$a = (x \times \frac{30}{u}) + ((v - y) \times \frac{30}{v}) + (y \times \frac{20}{u}) + ((v - x) \times \frac{20}{v})$	Distributed throughout
Mask 3	$a = ((u - x) \times \frac{30}{u}) + (y \times \frac{30}{v}) + ((u - y) \times \frac{20}{u}) + (y \times \frac{20}{v})$	Focused on the top right corner
Mask 4	$a = (x \times \frac{30}{u}) + (y \times \frac{30}{v}) + (x \times \frac{20}{u}) + (y \times \frac{20}{v})$	Focused on the bottom right corner
Mask 5	$a = \text{abs}(16 - x) \times \text{abs}(16 - y)$	The curved diamond shape
Mask 6 Mask 7 Mask 8	$a = 144 - \text{abs}(16 - x) \times \text{abs}(16 - y)$ $a = 100 - \text{abs}(16 - x) \times \text{abs}(16 - y)$ $a = 50 - \text{abs}(16 - x) \times \text{abs}(16 - y)$	Circular perturbation with different radius at centre
Mask 9	if ($0 \leq y \leq 5$ or $10 \leq y \leq 15$ or $20 \leq y \leq 25$ or $30 \leq y \leq 32$) : $a = \text{Mask 1}$ else : $a = -\text{Mask 2}$	A pattern of vertical lines
Mask 10	if ($0 \leq x \leq 5$ or $10 \leq x \leq 15$ or $20 \leq x \leq 25$ or $30 \leq x \leq 32$) : $a = \text{Mask 1}$ else : $a = -\text{Mask 2}$	A pattern of horizontal lines
Mask 11	if ($x \leq 16$ and $y \leq 16$) : $a = \text{Mask 1}$ if ($x \leq 16$ and $y > 16$) : $a = \text{Mask 2}$ if ($x > 16$ and $y \leq 16$) : $a = \text{Mask 3}$ if ($x > 16$ and $y > 16$) : $a = \text{Mask 4}$	Differs for different quadrants of the image
Mask 12	if ($x \leq 16$ and $y \leq 16$) : $a = \text{Mask 1}$ if ($x \leq 16$ and $y > 16$) : $a = \text{Mask 2}$ if ($x > 16$ and $y \leq 16$) : $a = -\text{Mask 3}$ if ($x > 16$ and $y > 16$) : $a = -\text{Mask 4}$	Differs for different quadrants of the image and produces a pattern effect of vertical lines

Non-Uniform Illumination Attack

Algorithm 1 Proposed NUI Attack Algorithm

Input: Image data $I \in \mathbb{R}^{u \times v}$ as input, Attack Mask ID $i \in \{1, 2, \dots, 12\}$, and perturbation weight value $k \in \{-2.2, -2.0, \dots, 2.0, 2.2\}$

Output: NUI Attacked Image $I_{M_i, k}$.

- 1: Generate the i^{th} Mask using Table 1 as
 $M_i = a(x, y), \forall x \in \{1, u\}$ and $\forall y \in \{1, v\}$.
 - 2: Perform the mask weighting with weight k as
 $M_{i, k} = M_i \times k$.
 - 3: Attack the image (I) to generate the perturbed image ($I_{M_i, k}$) as
 $I_{M_i, k} = I + M_{i, k}$.
-

Non-Uniform Illumination Attack



Figure 2: 1st column in the figure contains original images. The 2nd to 13th columns contain the images perturbed using mask 1st to 12th, respectively. The images are taken from CIFAR10, TinyImageNet, and CalTech256 datasets.

Non-Uniform Illumination Attack

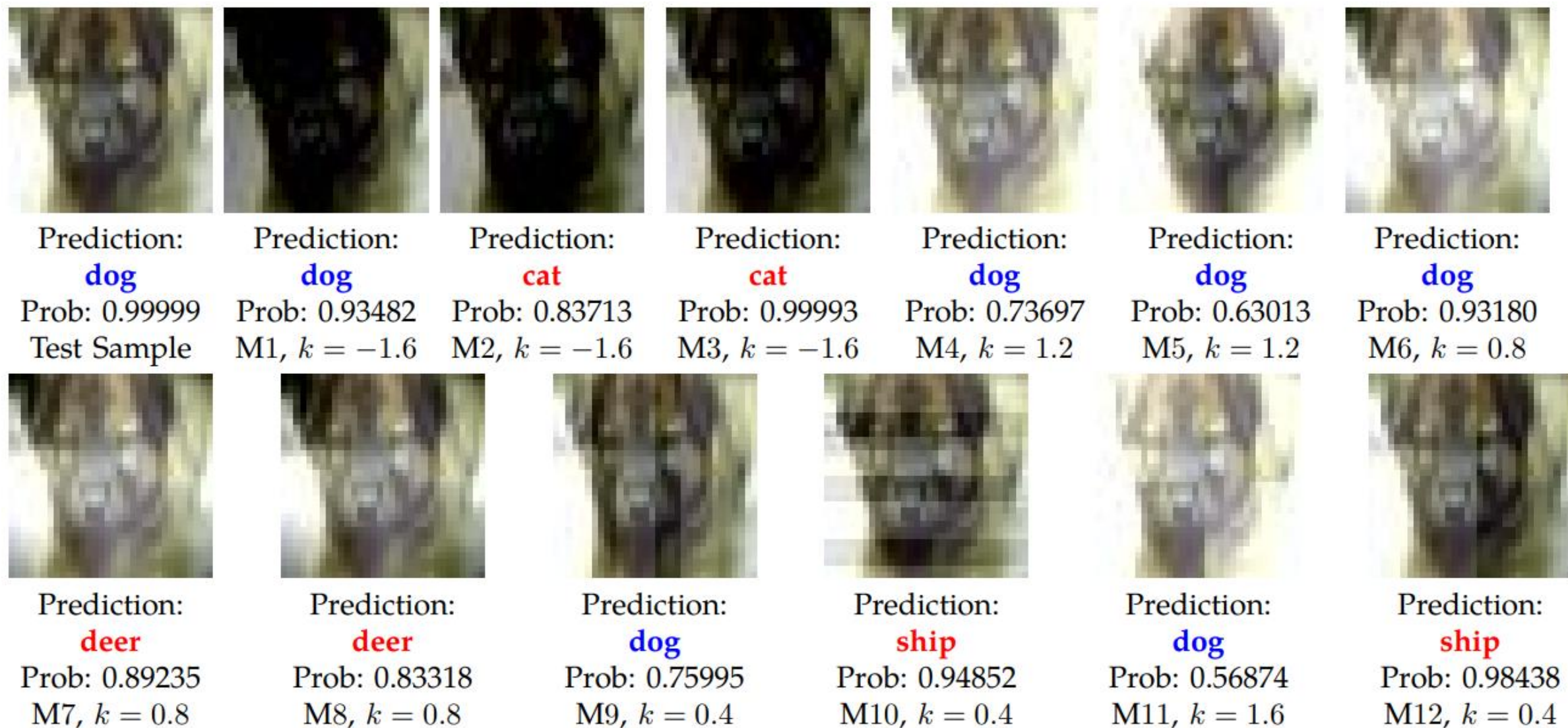


Figure 4: Predictions of ResNet18 for an original test image and NUI attacked images using different masks with varying weight (k). 'Prob' refers to probability and M_i refers to i^{th} mask.

Non-Uniform Illumination Attack

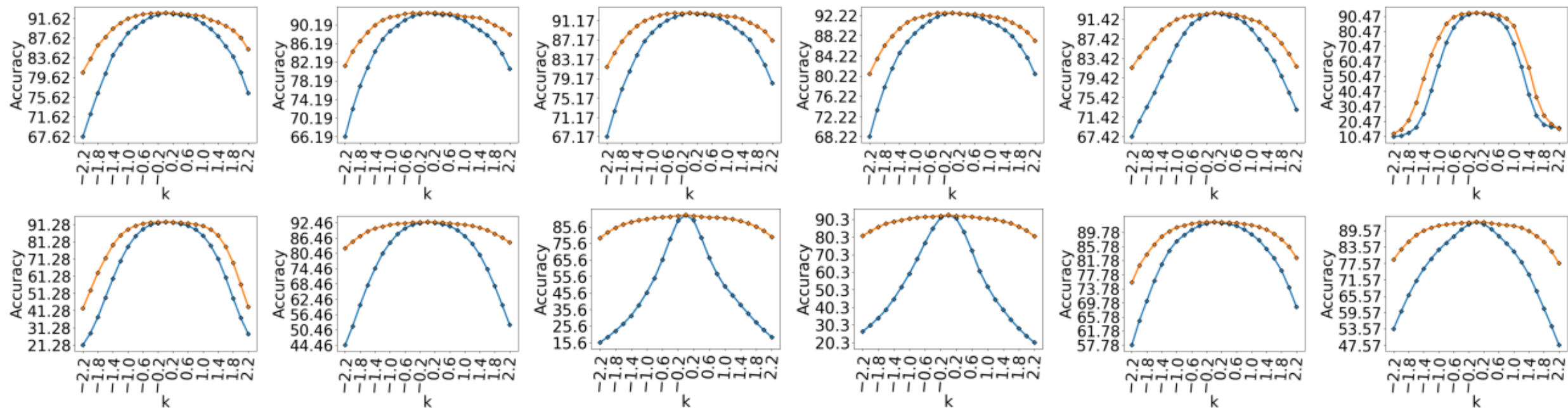


Figure 5: Results of VGG16 model on CIFAR10 dataset under different NUI attacks (test set). Blue and orange curves show the performance of the model trained on the original training set and the NUI perturbed training set, respectively.

Non-Uniform Illumination Attack

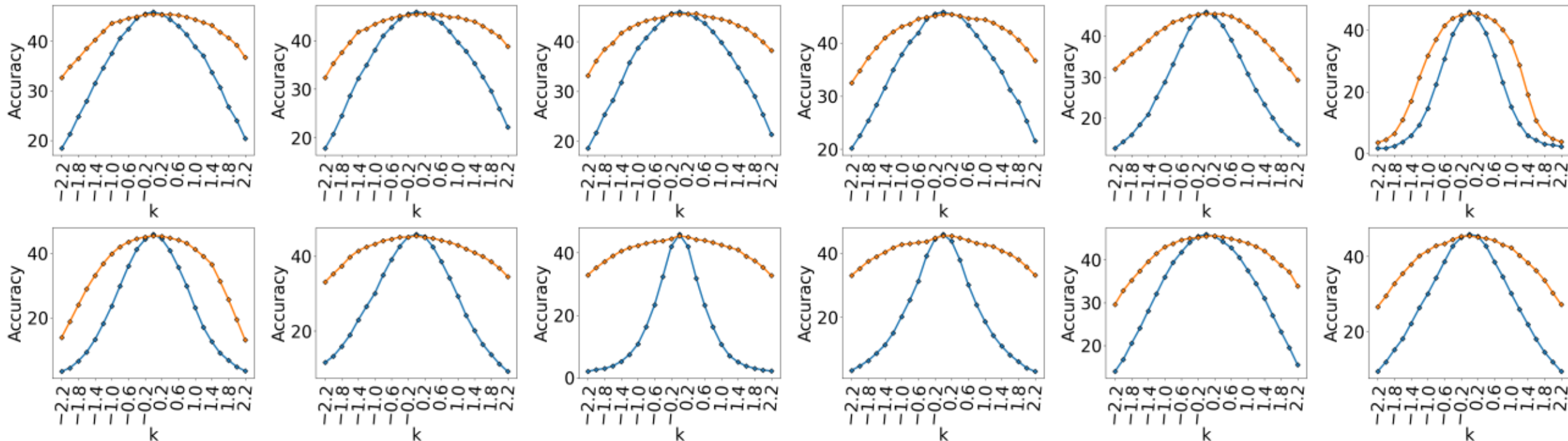


Figure 7: Results of VGG16 model on TinyImageNet dataset under different NUI attacks (test set). Blue and orange curves show the results of the model trained on the original training set and the NUI perturbed training set, respectively.

Non-Uniform Illumination Attack

Table II: The % decrease in the accuracy of models on various test sets under NUI attack. Results are reported for $k = -1.4$

Model	Dataset	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
VGG16	CIFAR10	9.19	8.91	9.33	8.5	13.9	72.29	35.47	19.63	65.63	51.44	12.93	18.49
VGG16	TinyImageNet	31.22	29.74	30.7	31.09	54.4	87.3	71.17	49.9	88.61	75.31	38.72	51.72
VGG19	CalTech256	22.12	21.56	21.66	21.17	43.86	83.8	61.71	39.58	87.9	78.26	29.45	43.02
ResNet18	CIFAR10	8.55	7.23	7.4	7.2	13.21	66.7	32.07	9.75	54.65	50.93	11	13.55
ResNet18	TinyImageNet	32.53	31.4	30.01	30.67	53.67	89.89	73.89	53.29	85.43	72.57	39.08	46.97
ResNet18	CalTech256	28.47	27.09	28.04	25.76	47.17	86.72	67.27	43.75	87.57	81.78	36.17	46.84
Mobilenet	CIFAR10	15.10	14.22	13.84	14.23	17.04	73.59	43.63	13.07	61.91	59.87	19.21	26.87
Inception	CIFAR10	8.45	7.79	8.02	7.99	11.00	68.02	30.10	7.48	63.24	54.98	11.63	18.32

Table III: The % increase in the accuracy of models on various test sets, after applying the proposed defense technique. Results are reported for $k = -1.4$

Model	Dataset	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
VGG16	CIFAR10	6.36	6.73	7.14	5.7	11.9	91	32.65	21.17	177.3	97.66	9.77	18.4
VGG16	TinyImageNet	27.64	29.73	31	19.74	85.83	193.7	150.5	80.66	675.8	256	40.8	71
VGG19	CalTech256	18.81	19.4	19.27	16.8	55.21	149.4	98.25	51.18	632.7	304.2	28.92	51.87
ResNet18	CIFAR10	4.11	3.59	3.75	3.75	9.42	68.7	26.18	7.12	108.8	93.73	6.82	9.97
ResNet18	TinyImageNet	30.4	30.26	26.84	26.92	80.85	262.4	169.2	90	503.4	215.8	41.82	63.62
ResNet18	CalTech256	33.71	31.85	33.81	29	77	237	144.1	70.44	664.1	418.5	48.5	79.98
Mobilenet	CIFAR10	13.04	12.95	12.55	12.65	14.58	109.65	54.09	12.26	147.37	136.82	18.42	27.81
Inception	CIFAR10	5.21	5.13	7.62	4.8	6.86	98.86	27.44	5.02	157.46	111.51	7.99	16.32

Non-Uniform Illumination Attack

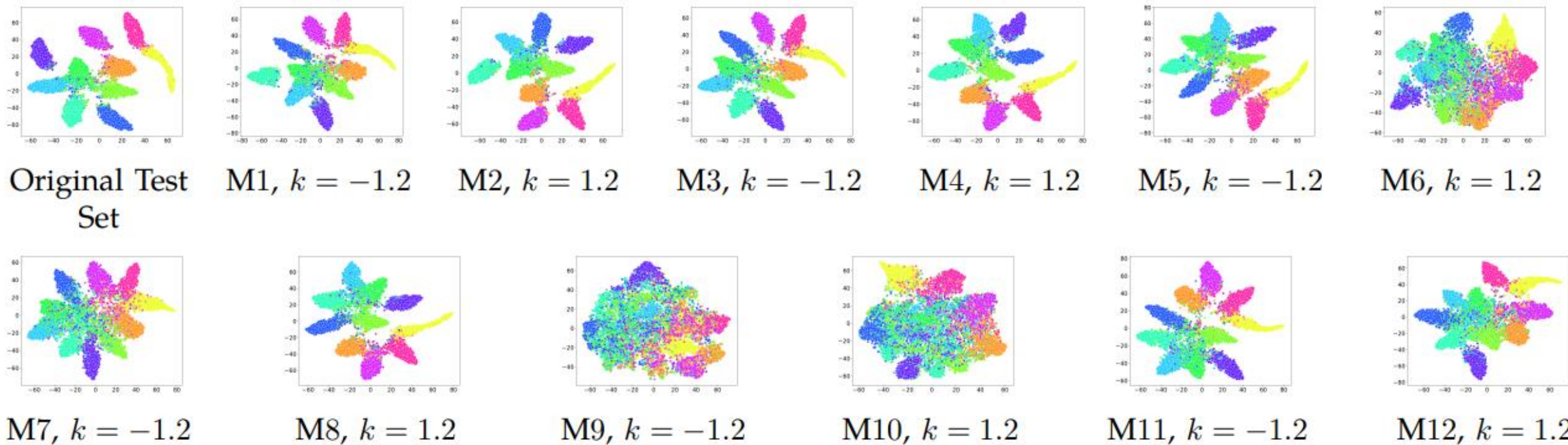


Figure 13: Shows the t-SNE plot on CIFAR10 test set using InceptionV3 model for 12 masks and 1.2 and -1.2 values of (k) . Here, M_i refers to i^{th} mask.

Adversarial examples: Outline

- Generating adversarial examples
 - Finding smallest “fooling” transformation
 - Gradient ascent
 - Fast gradient sign, iterative variants
 - Universal adversarial perturbations
- Why are neural networks easy to fool?
- Defending against adversarial examples
 - Adversarial training
 - Learning to reject adversarial examples
 - Robust architectures
 - Image pre-processing
- Recent Works
- “Open” topics
 - Broadening the scope of adversarial examples
 - Adversarial examples and human perception

Adversarial examples for detection

- It is much harder to fool a detector like Faster R-CNN or YOLO than a classifier; larger perturbations are required



Adversarial examples for detection

- It is much harder to fool a detector like Faster R-CNN or YOLO than a classifier; larger perturbations are required
- It is even harder to fool a detector with physical objects



"All three patterns reliably fool detectors when mapped into videos. However, physical instances of these patterns are not equally successful. The first two stop signs, as physical objects, only occasionally fool Faster RCNN; the third one, which has a much more extreme pattern, is more effective."

Robust adversarial examples

3D printed adversarial object ([YouTube video](#))

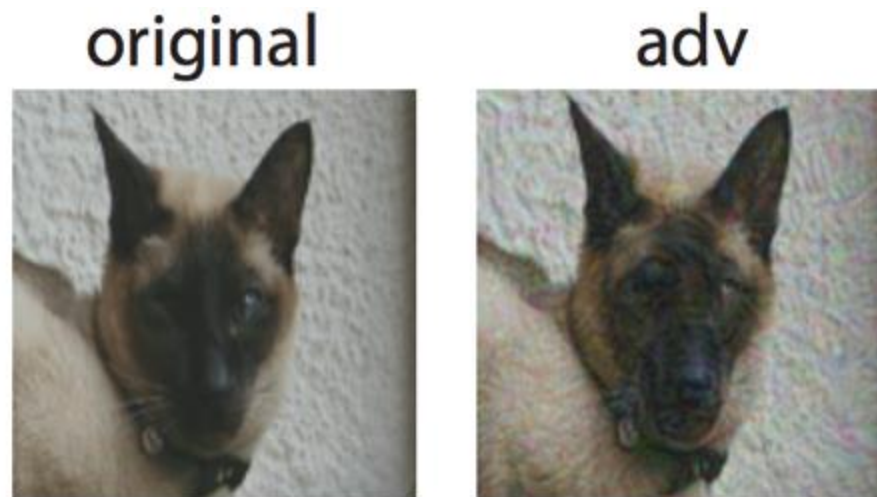


■ classified as turtle ■ classified as rifle
■ classified as other

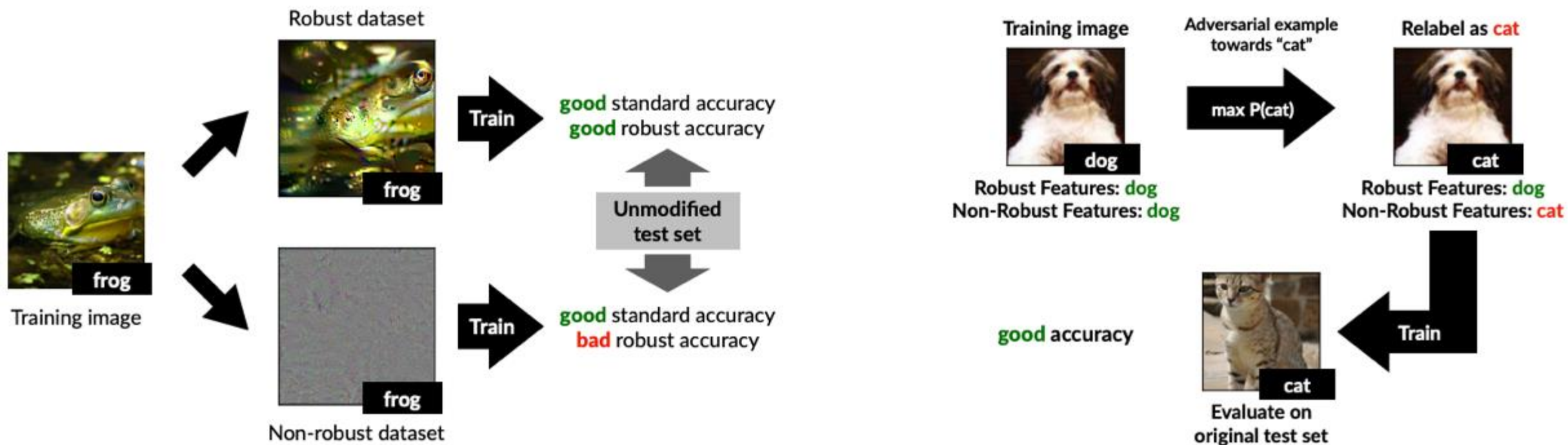
<https://blog.openai.com/robust-adversarial-inputs/>

Adversarial examples and humans

- Adversarial examples that are designed to transfer across multiple architectures can also be shown to confuse the human visual system in rapid presentation settings



Adversarial examples are not bugs, they are features



Disentangle features into robust and non-robust

Construct a dataset which appears mislabeled to humans (via adversarial examples) but results in good accuracy on the original test set

ACKNOWLEDGEMENT

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- NVDIEA Deep Learning Teaching Kit
- And many more...