Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

October 30, 2024

Today's Agenda

1 Universal Adversarial Perturbation (UAP)

2 Adversarial Patch

Universal Adversarial Perturbation (UAP)

Universal Adversarial Perturbation

Universal adversarial perturbations

Seyed-Mohsen Moosavi-Dezfooli*†
seyed.moosavi@epfl.ch

Alhussein Fawzi*†
alhussein.fawzi@epfl.ch

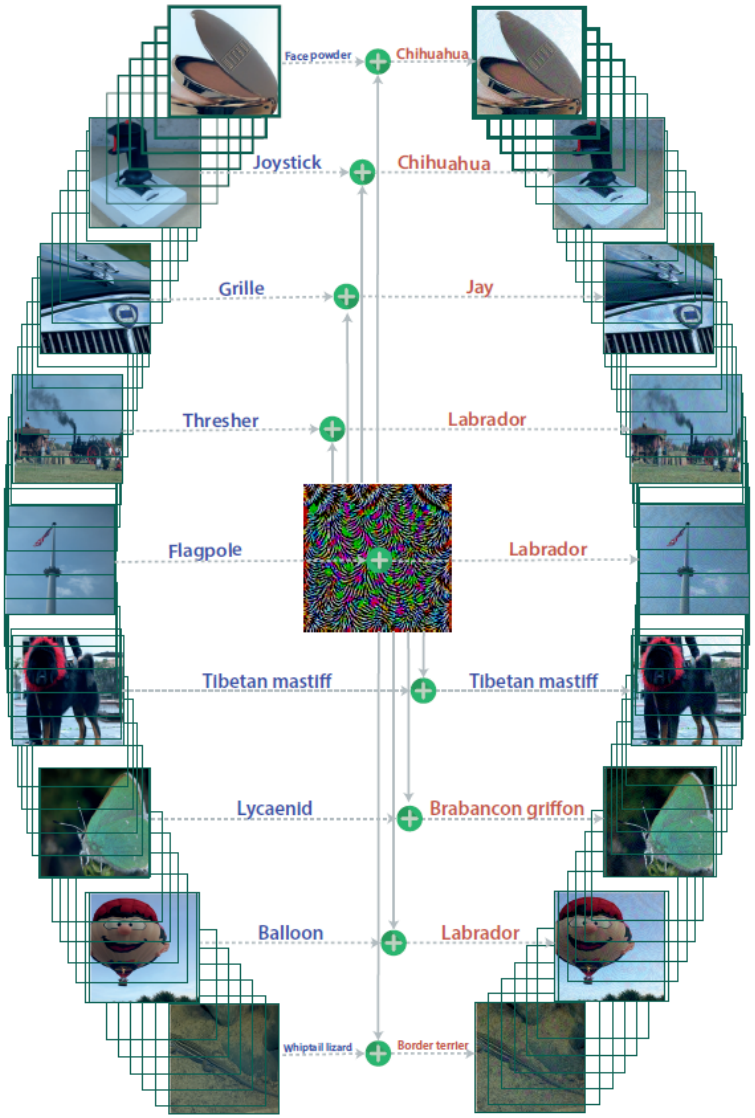
Omar Fawzi‡
omar.fawzi@ens-lyon.fr

Pascal Frossard†
pascal.frossard@epfl.ch

Abstract

- We show the existence of a **universal (image-agnostic)** and very small perturbation vector that causes natural images to be misclassified with high probability.
  - They generalize very well across neural networks.
- The surprising existence of universal perturbations reveals important **geometric correlations among the high-dimensional decision boundary** of classifiers.

Universal (Image-agnostic) Perturbation



Universal perturbations

Let μ denote a distribution of images in $\mathbb{R}^d$, and $\hat{k}$ define a classification function that outputs for each image $x \in \mathbb{R}^d$ an estimated label $\hat{k}(x)$. Let $X = \{x_1, ..., x_m\}$ be a set of images sampled from the distribution μ.

Universal perturbations

Let μ denote a distribution of images in $\mathbb{R}^d$, and $\hat{k}$ define a classification function that outputs for each image $x \in \mathbb{R}^d$ an estimated label $\hat{k}(x)$. Let $X = \{x_1, ..., x_m\}$ be a set of images sampled from the distribution μ.

The main focus of this paper is to seek perturbation vectors $v \in \mathbb{R}^d$ that fool the classifier $\hat{k}$ on almost all data points sampled from μ. That is, we seek a vector v such that

$$\hat{k}(x + v) \neq \hat{k}(x) \text{ for } \textbf{most } x \sim \mu$$

Universal perturbations

Let μ denote a distribution of images in $\mathbb{R}^d$, and $\hat{k}$ define a classification function that outputs for each image $x \in \mathbb{R}^d$ an estimated label $\hat{k}(x)$. Let $X = \{x_1, ..., x_m\}$ be a set of images sampled from the distribution μ.

The main focus of this paper is to seek perturbation vectors $v \in \mathbb{R}^d$ that fool the classifier $\hat{k}$ on almost all data points sampled from μ. That is, we seek a vector v such that

$$\hat{k}(x + v) \neq \hat{k}(x) \text{ for } \textbf{most } x \sim \mu$$

The goal is to find v that satisfies the following two constraints:

1. $\|v\|_P \leq \xi$
2. $\underset{x \sim \mu}{\mathbb{P}} \left(\hat{k}(x + v) \neq \hat{k}(x) \right) \geq 1 - \delta$

The parameter ξ controls the magnitude of the perturbation vector v, and δ quantifies the desired fooling rate for all images sampled from the distribution μ.

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,
  - If the current universal perturbation v does not fool $\hat{k}$ for data point xi, we seek the extra perturbation Δv_i with minimal norm that allows to fool $\hat{k}$ for data point x_i by solving the following optimization problem:

$$\Delta v_i \leftarrow \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,
  - If the current universal perturbation v does not fool $\hat{k}$ for data point xi, we seek the extra perturbation Δv_i with minimal norm that allows to fool $\hat{k}$ for data point x_i by solving the following optimization problem:

$$\Delta v_i \leftarrow \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

  - To ensure that the constraint $\|v\|_P \leq \xi$ is satisfied, the updated universal perturbation is further projected on the L_P ball of radius ξ and centered at 0. That is, let $\mathcal{P}_{P,\xi}$ be the projection operator defined as follows:

$$\mathcal{P}_{P,\xi}(v) = \underset{v'}{argmin} \|v - v'\|_2 \quad \text{subject to } \|v'\|_P \leq \xi.$$

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,
  - If the current universal perturbation v does not fool $\hat{k}$ for data point xi, we seek the extra perturbation Δv_i with minimal norm that allows to fool $\hat{k}$ for data point x_i by solving the following optimization problem:

$$\Delta v_i \leftarrow \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

  - To ensure that the constraint $\|v\|_P \leq \xi$ is satisfied, the updated universal perturbation is further projected on the L_P ball of radius ξ and centered at 0. That is, let $\mathcal{P}_{P,\xi}$ be the projection operator defined as follows:

$$\mathcal{P}_{P,\xi}(v) = \underset{v'}{argmin} \|v - v'\|_2 \quad \text{subject to } \|v'\|_P \leq \xi.$$

  - Then, our update rule is given by $v \leftarrow \mathcal{P}_{P,\xi}(v + \Delta v_i)$.

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,
  - If the current universal perturbation v does not fool $\hat{k}$ for data point xi, we seek the extra perturbation Δv_i with minimal norm that allows to fool $\hat{k}$ for data point x_i by solving the following optimization problem:

$$\Delta v_i \leftarrow \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

  - To ensure that the constraint $\|v\|_P \leq \xi$ is satisfied, the updated universal perturbation is further projected on the L_P ball of radius ξ and centered at 0. That is, let $\mathcal{P}_{P,\xi}$ be the projection operator defined as follows:

$$\mathcal{P}_{P,\xi}(v) = \underset{v'}{argmin} \|v - v'\|_2 \quad \text{subject to } \|v'\|_P \leq \xi.$$

  - Then, our update rule is given by $v \leftarrow \mathcal{P}_{P,\xi}(v + \Delta v_i)$.
- Several passes on the data set X are performed to improve the quality of the universal perturbation.

UAP Algorithm

- Algorithm 1 proceeds iteratively over the data points in X and gradually builds the universal perturbation.
- At each iteration,
  - If the current universal perturbation v does not fool $\hat{k}$ for data point xi, we seek the extra perturbation Δv_i with minimal norm that allows to fool $\hat{k}$ for data point x_i by solving the following optimization problem:

$$\Delta v_i \leftarrow \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

  - To ensure that the constraint $\|v\|_P \leq \xi$ is satisfied, the updated universal perturbation is further projected on the L_P ball of radius ξ and centered at 0. That is, let $\mathcal{P}_{P,\xi}$ be the projection operator defined as follows:

$$\mathcal{P}_{P,\xi}(v) = \underset{v'}{argmin} \|v - v'\|_2 \quad \text{subject to } \|v'\|_P \leq \xi.$$

  - Then, our update rule is given by $v \leftarrow \mathcal{P}_{P,\xi}(v + \Delta v_i)$.
- Several passes on the data set X are performed to improve the quality of the universal perturbation.
- The algorithm is terminated when the empirical "fooling rate" on the perturbed data set $X_v := \{x_1 + v, ..., x_m + v\}$ exceeds the target threshold $1 - \delta$. That is, we stop the algorithm whenever

$$Err(X_v) := \frac{1}{m} \sum_{i=1}^{m} 1_{\hat{k}(x_i+v) \neq \hat{k}(x_i)} \geq 1 - \delta.$$

UAP Algorithm

Algorithm 1 Computation of universal perturbations.

input: Data points X, classifier $\hat{k}$, desired ℓ_p norm of the perturbation ξ, desired accuracy on perturbed samples δ.
output: Universal perturbation vector v.
Initialize $v \leftarrow 0$.
while $\text{Err}(X_v) \leq 1 - \delta$ **do**
    for each datapoint $x_i \in X$ **do**
        if $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
            Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

            Update the perturbation:

$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

        end if
    end for
end while

UAP Algorithm Properties

- Universal perturbation is **class-agnostic**
- Interestingly, in practice, the number of data points $|X| = m$ need not be large to compute a universal perturbation.
  - we can set m to be much smaller than the number of training points.

UAP Algorithm Properties

- Universal perturbation is **class-agnostic**
- Interestingly, in practice, the number of data points $|X| = m$ need not be large to compute a universal perturbation.
  - we can set m to be much smaller than the number of training points.

- L-BFGS, FGSM, PGD, or DeepFool attacks can be used to compute Δv_i.

UAP Algorithm Properties

- Universal perturbation is **class-agnostic**
- Interestingly, in practice, the number of data points $|X| = m$ need not be large to compute a universal perturbation.
  - we can set m to be much smaller than the number of training points.
- L-BFGS, FGSM, PGD, or DeepFool attacks can be used to compute Δv_i.
- Different random shuffling of the set X naturally lead to **a diverse set of universal perturbations** v satisfying the required constraints.

Evaluation

The table reports the fooling ratio, that is the proportion of images that change labels when perturbed by universal perturbation.

- ILSVRC2012 dataset
- Results are reported for $P = 2$ and $P = \infty$, where we respectively set $\xi = 2000$ and $\xi = 10/255$.
- Each result is reported on the set X, which is used to compute the perturbation, as well as on the 50000 validation images (that is not used in the process of the computation of the universal perturbation).

		CaffeNet [8]	VGG-F [2]	VGG-16 [17]	VGG-19 [17]	GoogLeNet [18]	ResNet-152 [6]
ℓ_2	X	85.4%	85.9%	90.7%	86.9%	82.9%	89.7%
	Val.	85.6	87.0%	90.3%	84.5%	82.0%	88.5%
ℓ_∞	X	93.1%	93.8%	78.5%	77.8%	80.8%	85.4%
	Val.	93.3%	93.7%	78.3%	77.8%	78.9%	84.0%

Table 1: Fooling ratios on the set X, and the validation set.

Evaluation

The table reports the fooling ratio, that is the proportion of images that change labels when perturbed by universal perturbation.

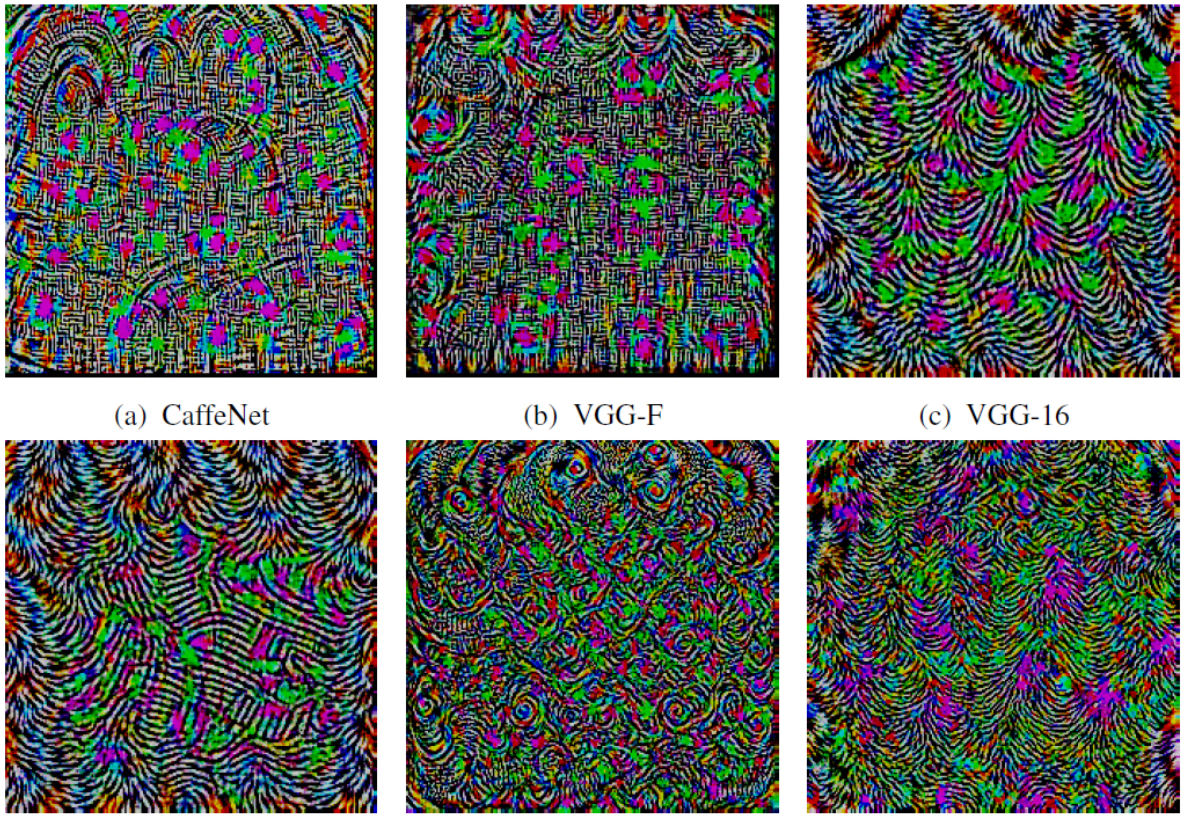
- ILSVRC2012 dataset
- Results are reported for $P = 2$ and $P = \infty$, where we respectively set $\xi = 2000$ and $\xi = 10/255$.
- Each result is reported on the set X, which is used to compute the perturbation, as well as on the 50000 validation images (that is not used in the process of the computation of the universal perturbation).

		CaffeNet [8]	VGG-F [2]	VGG-16 [17]	VGG-19 [17]	GoogLeNet [18]	ResNet-152 [6]
ℓ_2	X	85.4%	85.9%	90.7%	86.9%	82.9%	89.7%
	Val.	85.6	87.0%	90.3%	84.5%	82.0%	88.5%
ℓ_∞	X	93.1%	93.8%	78.5%	77.8%	80.8%	85.4%
	Val.	93.3%	93.7%	78.3%	77.8%	78.9%	84.0%

Table 1: Fooling ratios on the set X, and the validation set.

- These results have an element of surprise, as they show the existence of **single universal perturbation vectors that cause natural images to be misclassified with high probability**, albeit being quasi-imperceptible to humans.

Evaluation



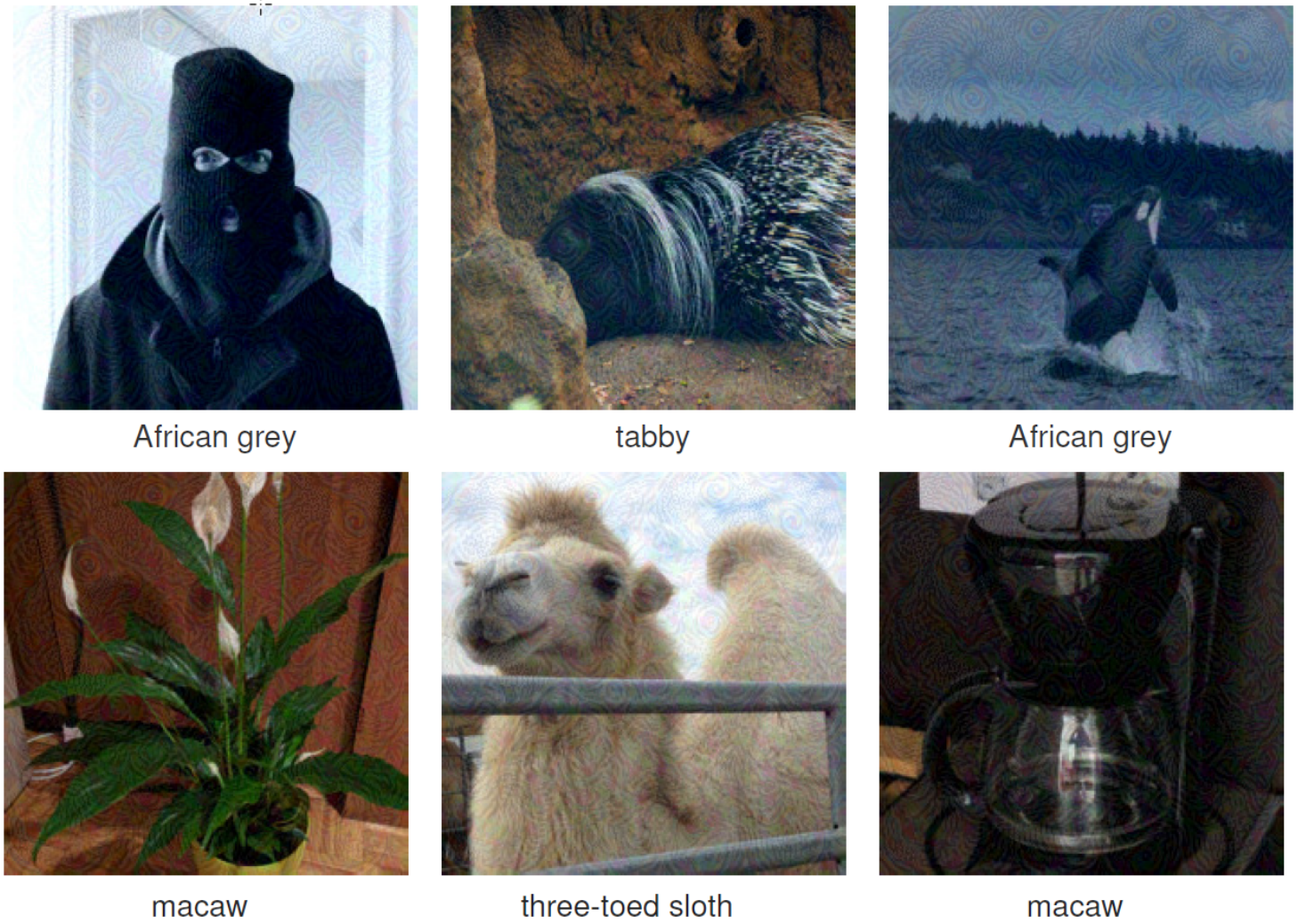
(a) CaffeNet (b) VGG-F (c) VGG-16

(d) VGG-19 (e) GoogLeNet (f) ResNet-152

Figure 4: Universal perturbations computed for different deep neural network architectures. Images generated with $p = \infty$, $\xi = 10$. The pixel values are scaled for visibility.

Evaluation

- Attack on GoogLeNet



Examples of perturbed images and their corresponding labels.

Cross-model Universality (doubly-universal)

	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%**

Table 2: Generalizability of the universal perturbations across different networks. The percentages indicate the fooling rates. The rows indicate the architecture for which the universal perturbations is computed, and the columns indicate the architecture for which the fooling rate is reported.

Diversity of universal perturbations

Note that the normalized inner products for any pair of universal perturbations does not exceed 0.1, which highlights the diversity of such perturbations.

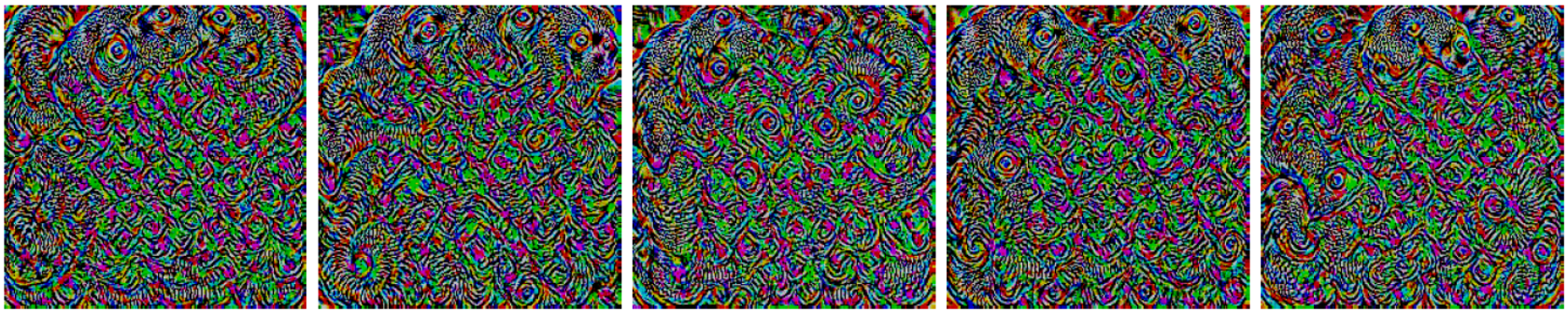


Figure 5: Diversity of universal perturbations for the GoogLeNet architecture. The five perturbations are generated using different random shufflings of the set X. Note that the normalized inner products for any pair of universal perturbations does not exceed 0.1, which highlights the diversity of such perturbations.

Evaluation

Figure 6 shows the fooling rates obtained on the validation set for different sizes of X for GoogLeNet.

- Note for example that with a set X containing only 500 images, we can fool more than 30% of the images on the validation set.
- This result is significant when compared to the number of classes in ImageNet (1000), as it shows that we can **fool a large set of unseen images**, even when using a set X containing **less than one image per class!**

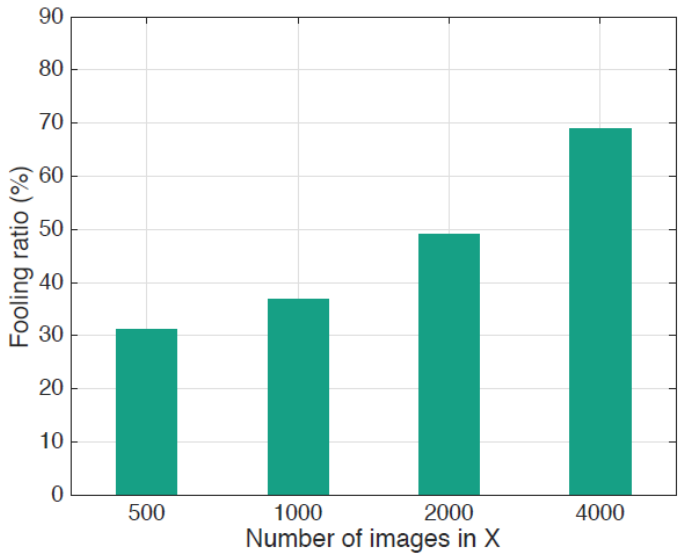


Figure 6: Fooling ratio on the validation set versus the size of X. Note that even when the universal perturbation is computed on a very small set X (compared to training and validation sets), the fooling ratio on validation set is large.

Visualization of the Effect of Universal Perturbations

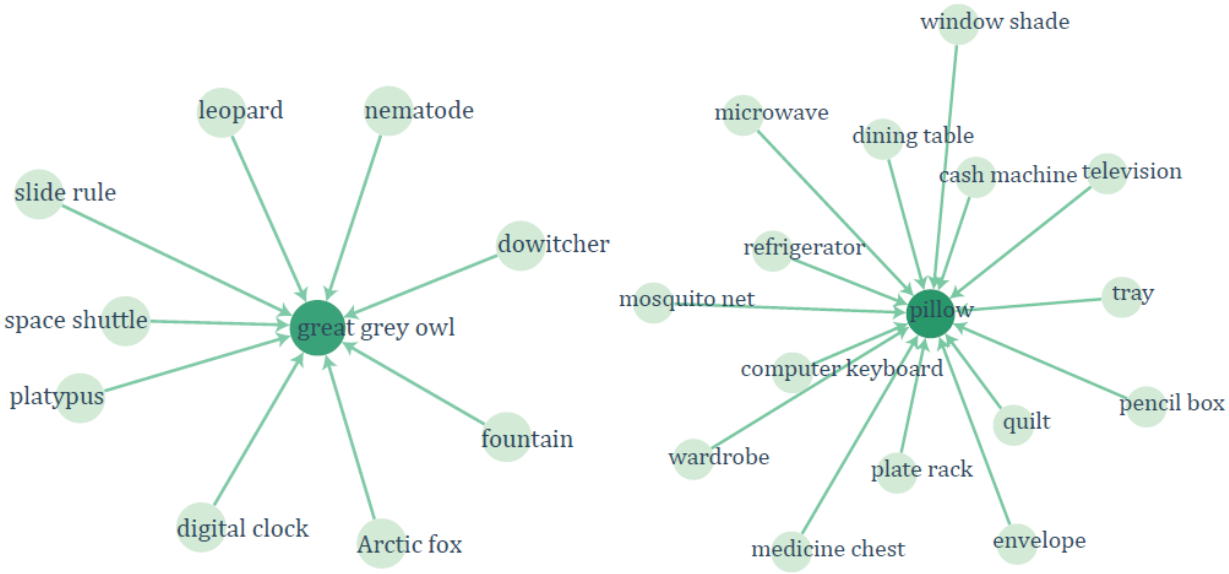


Figure 7: Two connected components of the graph $G = (V, E)$, where the vertices are the set of labels, and directed edges $i \to j$ indicate that most images of class i are fooled into class j.

Explaining the vulnerability to universal perturbations

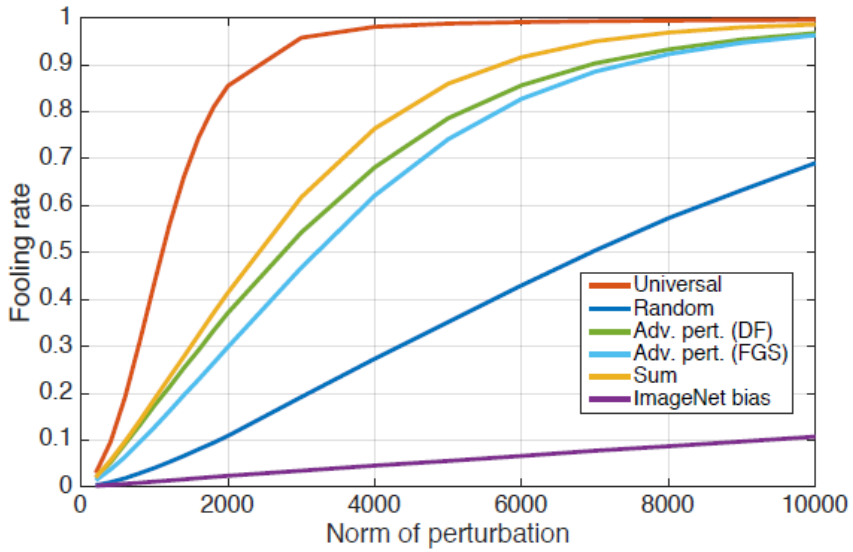
We compare universal perturbations with other types of perturbations

- Random perturbation
- Adversarial perturbation computed for a randomly picked sample (computed using the DeepFool and FGSM)
- Sum of adversarial perturbations over X
- Mean of the images (or ImageNet bias).

Explaining the vulnerability to universal perturbations

We compare universal perturbations with other types of perturbations

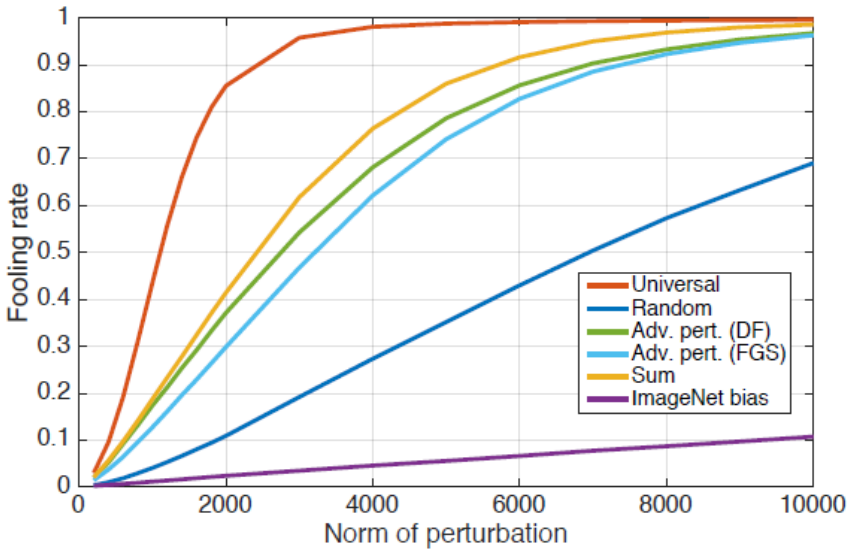
- Random perturbation
- Adversarial perturbation computed for a randomly picked sample (computed using the DeepFool and FGSM)
- Sum of adversarial perturbations over X
- Mean of the images (or ImageNet bias).



Explaining the vulnerability to universal perturbations

We compare universal perturbations with other types of perturbations

- Random perturbation
- Adversarial perturbation computed for a randomly picked sample (computed using the DeepFool and FGSM)
- Sum of adversarial perturbations over X
- Mean of the images (or ImageNet bias).



The large difference between universal and random perturbations suggests that the universal perturbation exploits some **geometric correlations between different parts of the decision boundary** of the classifier.

- If the orientations of the decision boundary in the neighborhood of different data points were completely uncorrelated, the norm of the best universal perturbation would be comparable to that of a random perturbation.

Explaining the vulnerability to universal perturbations

For each image x in the validation set, we compute the adversarial perturbation vector

$$r(x) = \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x+r) \neq \hat{k}(x).$$

To quantify the correlation between different regions of the decision boundary of the classifier, we define the matrix

$$N = \left[\frac{r(x_1)}{\|r(x_1)\|_2} \cdots \frac{r(x_n)}{\|r(x_n)\|_2} \right]$$

of normal vectors to the decision boundary in the vicinity of n data points in the validation set.

Explaining the vulnerability to universal perturbations

For each image x in the validation set, we compute the adversarial perturbation vector

$$r(x) = \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x+r) \neq \hat{k}(x).$$

To quantify the correlation between different regions of the decision boundary of the classifier, we define the matrix

$$N = \left[\frac{r(x_1)}{\|r(x_1)\|_2} \cdots \frac{r(x_n)}{\|r(x_n)\|_2} \right]$$

of normal vectors to the decision boundary in the vicinity of n data points in the validation set.

- For binary linear classifiers, the decision boundary is a hyperplane, and N is of rank 1, as all normal vectors are collinear.

Explaining the vulnerability to universal perturbations

For each image x in the validation set, we compute the adversarial perturbation vector

$$r(x) = \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x+r) \neq \hat{k}(x).$$

To quantify the correlation between different regions of the decision boundary of the classifier, we define the matrix

$$N = \left[\frac{r(x_1)}{\|r(x_1)\|_2} \cdots \frac{r(x_n)}{\|r(x_n)\|_2} \right]$$

of normal vectors to the decision boundary in the vicinity of n data points in the validation set.

- For binary linear classifiers, the decision boundary is a hyperplane, and N is of rank 1, as all normal vectors are collinear.
- To capture more generally the correlations in the decision boundary of complex classifiers, we compute the singular values of the matrix N.

Explaining the vulnerability to universal perturbations

For each image x in the validation set, we compute the adversarial perturbation vector

$$r(x) = \underset{r}{argmin} \|r\|_2 \quad s.t.\ \hat{k}(x+r) \neq \hat{k}(x).$$

To quantify the correlation between different regions of the decision boundary of the classifier, we define the matrix

$$N = \left[\frac{r(x_1)}{\|r(x_1)\|_2} \cdots \frac{r(x_n)}{\|r(x_n)\|_2} \right]$$

of normal vectors to the decision boundary in the vicinity of n data points in the validation set.

- For binary linear classifiers, the decision boundary is a hyperplane, and N is of rank 1, as all normal vectors are collinear.
- To capture more generally the correlations in the decision boundary of complex classifiers, we compute the singular values of the matrix N.
- We further show in the same figure the singular values obtained when the columns of N are sampled uniformly at random from the unit sphere.

Recall

Recall: Rank of a matrix

The rank of matrix A is the dimension of the vector space generated (or spanned) by its columns.

- This corresponds to the **maximal number of linearly independent columns** of A.
- The rank of A equals the number of non-zero singular values, which is the same as the number of non-zero diagonal elements in Σ in the singular value decomposition $A = U\Sigma V^T$

Recall

Recall: Singular Value Decomposition (SVD)

The singular value decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. Consider a matrix A_{mn}, it can be uniquely decomposed as

$$A_{mn} = U_{mm}\Sigma_{mn}V^T_{nn} = \sum_{i=1}^{r} u_i \sigma_i v_i^T$$

where U is an $m \times m$ matrix whose rows and columns are orthornormal (so $U^TU = UU^T = I_m$), V is $n \times n$ matrix whose rows and columns are orthonormal (so $V^TV = VV^T = I_n$), and Σ is a $m \times n$ matrix (a diagonal matrix) containing the $r = min(m, n)$ singular values $\sigma_i \geq 0$ on the main diagonal, with 0s filling the rest of the matrix.

The singular values are the diagonal entries of the Σ matrix and are arranged in descending order. The columns of U are the left singular vectors, and the columns of V are the right singular vectors. Since the rank of matrix $u_i\sigma_i v_i^T$ is 1 (all columns are a factor of u_i) and u_is are linearly independent, the rank of A equals the number of non-zero singular values.

Explaining the vulnerability to universal perturbations

Observe that, while the latter singular values have a slow decay, the singular values of N decay quickly, which confirms the existence of large correlations and redundancies in the decision boundary of deep networks.

- More precisely, this suggests the existence of a subspace $\mathcal{S}$ of low dimension d' (with $d' << d$), that contains most normal vectors to the decision boundary in regions surrounding natural images.

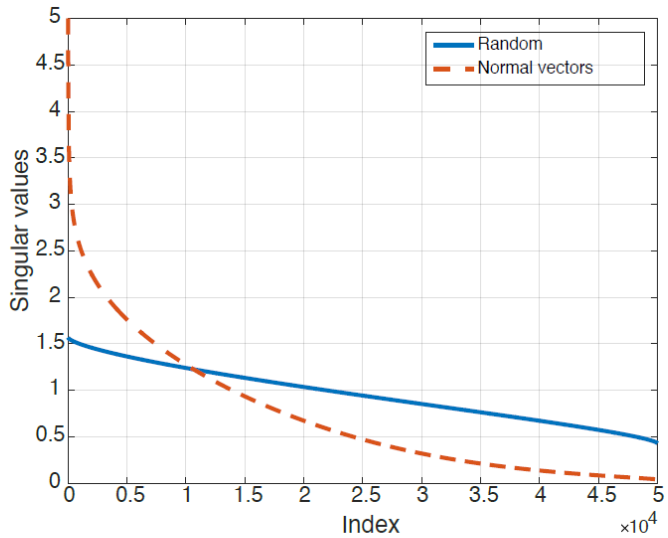


Figure 9: Singular values of matrix N containing normal vectors to the decision decision boundary.

Explaining the vulnerability to universal perturbations

We hypothesize that the existence of universal perturbations fooling most natural images is partly due to the existence of such a low-dimensional subspace that captures the correlations among different regions of the decision boundary.

- To verify this hypothesis, we choose a random vector of norm $\xi = 2000$ belonging to the subspace $\mathcal{S}$ spanned by the first **100 singular vectors**.
- Such a perturbation can fool nearly 38% of these images, thereby showing that a random direction in this well-sought subspace $\mathcal{S}$ significantly outperforms random perturbations (random perturbations can only fool 10% of the data).

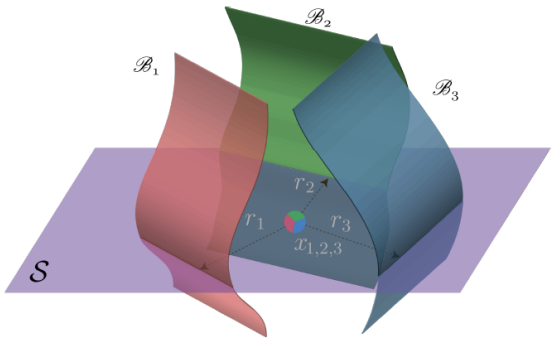
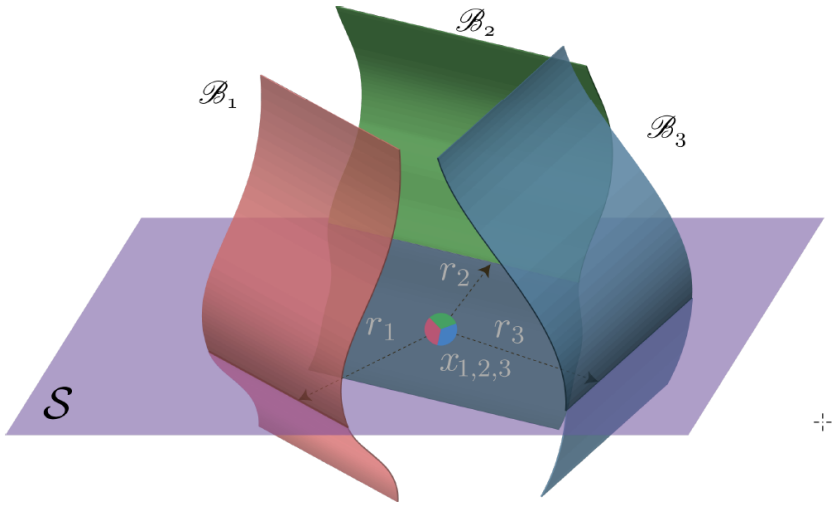


Figure 10: Illustration of the low dimensional subspace $\mathcal{S}$ containing normal vectors to the decision boundary in regions surrounding natural images. For the purpose of this illustration, we super-impose three data-points $\{x_i\}_{i=1}^3$, and the adversarial perturbations $\{r_i\}_{i=1}^3$ that send the respective datapoints to the decision boundary $\{\mathscr{B}_i\}_{i=1}^3$ are shown. Note that $\{r_i\}_{i=1}^3$ all live in the subspace $\mathcal{S}$.

Explaining the vulnerability to universal perturbations

We hypothesize that the existence of universal perturbations fooling most natural images is partly due to the existence of such a low-dimensional subspace that captures the correlations among different regions of the decision boundary.

- To verify this hypothesis, we choose a random vector of norm $\xi = 2000$ belonging to the subspace $\mathcal{S}$ spanned by the first **100 singular vectors**.
- Such a perturbation can fool nearly 38% of these images, thereby showing that a random direction in this well-sought subspace $\mathcal{S}$ significantly outperforms random perturbations (random perturbations can only fool 10% of the data).



Unlike the above experiment, the proposed algorithm does not choose a random vector in this subspace, but rather chooses a specific direction in order to maximize the overall fooling rate. This explains the gap between the fooling rates obtained with the random vector strategy in $\mathcal{S}$ and Algorithm 1.

Adversarial Patch

Adversarial Patch

Adversarial Patch

Tom B. Brown, Dandelion Mané,* Aurko Roy, Martín Abadi, Justin Gilmer
{tombrown,dandelion,aurkor,abadi,gilmer}@google.com

Abstract

- The paper presents a method to create **universal, robust, targeted adversarial image patches** in the real world.
  - Universal because they can be used to attack any scene
  - Robust because they work under a wide variety of transformations
  - Targeted because they can cause a classifier to output any target class.

- This work explores what is possible if an attacker is **no longer restricted to small or imperceptible perturbation**.

Adversarial Patch

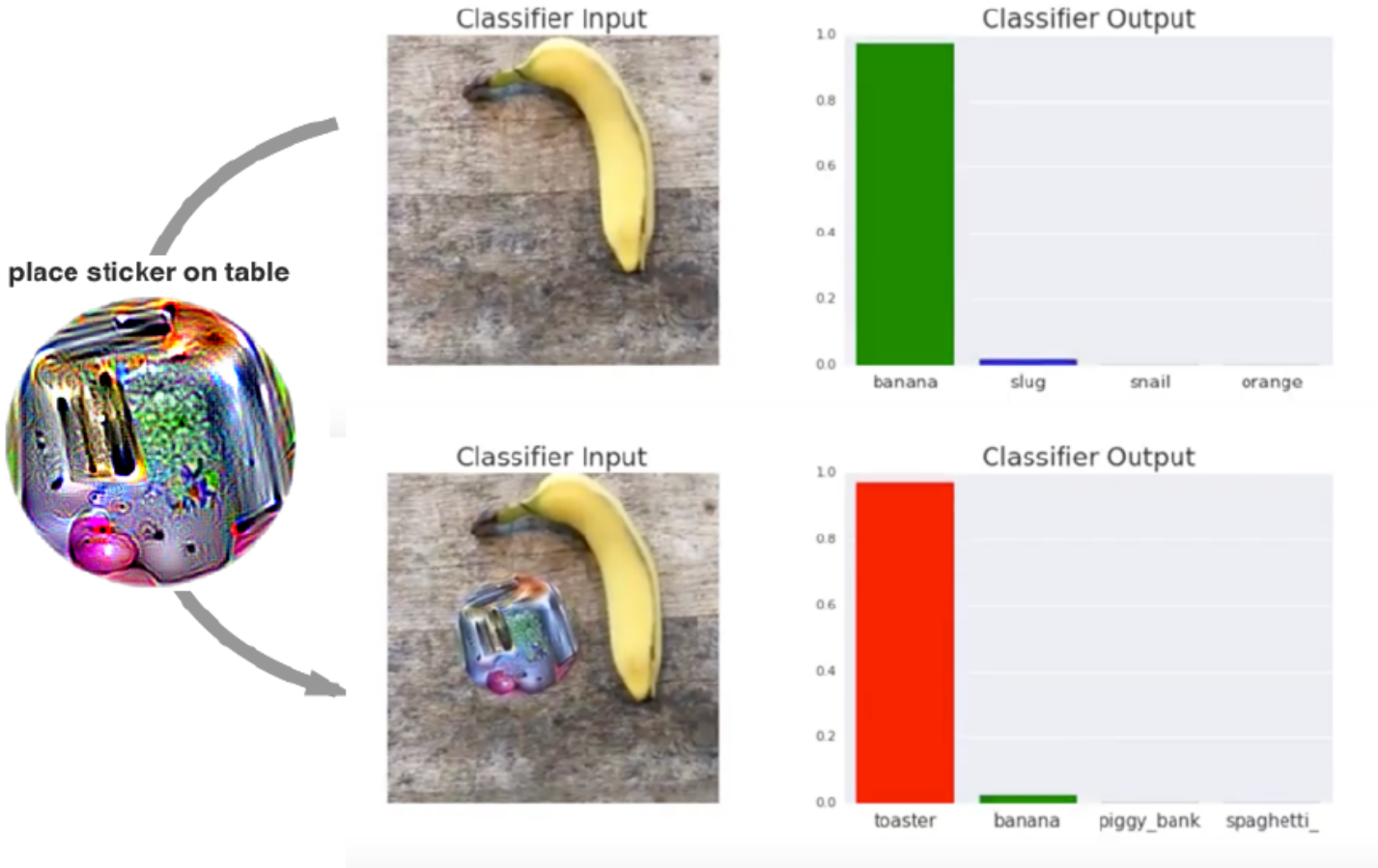


Figure 1: A real-world attack on VGG16, using a physical patch generated by the white-box ensemble method described in Section 3. When a photo of a tabletop with a banana and a notebook (top photograph) is passed through VGG16, the network reports class 'banana' with 97% confidence (top plot). If we physically place a sticker targeted to the class "toaster" on the table (bottom photograph), the photograph is classified as a toaster with 99% confidence (bottom plot). See the following video for a full demonstration: `https://youtu.be/i1sp4X57TL4`

The Approach

- The attack completely **replaces a part of the image with a patch**.
  - We mask the patch to allow it to take any shape, and then train over a variety of images, applying a random **translation, scaling, and rotation** on the patch in each image, optimizing using gradient descent.

The Approach

- The attack completely **replaces a part of the image with a patch**.
  - We mask the patch to allow it to take any shape, and then train over a variety of images, applying a random **translation, scaling, and rotation** on the patch in each image, optimizing using gradient descent.
- The patch is trained to optimize the objective function

$$\hat{p} = \underset{p}{argmax}\ \mathbb{E}_{x\sim X, t\sim T, l\sim L}[logPr(\hat{y}|A(p,x,l,t))]$$

where X is a training set of images, T is a distribution over transformations of the patch, L is a distribution over locations in the image, and $\hat{y}$ is the target class.

- Note that this expectation is over images, which encourages the trained patch to work regardless of what is in the background (perturbation is universal).