Adversarial Examples Are Not Bugs, They Are Features!

Alireza Sakhaeirad / Hamidreza Akbari

Department of Computer Engineering,
Sharif University of Technology

May, 2023

Motivation

What is this?
How did you find out?

Claim

- Adversarial vulnerability is a direct result of our models sensitivity to well-generalizing features in the data.
- But how are these well-generalizing features influence the model's learning?

Standard Training Objective

$$\theta = \underset{\theta}{\textit{argmin}}\, \mathbb{E}_{x,y\sim\mathcal{D}}[\mathbb{L}(x, y; \theta]$$

- Model learns what its instructed to!
- We never instructed it to learn robust features!

Contributions (I)

- A robustified version for robust classification. We demonstrate that it is possible to effectively remove non-robust features from a dataset. Concretely, we create a training set (semantically similar to the original) on which standard training yields good robust accuracy on the original, unmodified test set. This finding establishes that adversarial vulnerability is not necessarily tied to the standard training framework, but is also a property of the dataset.

Contributions (I)

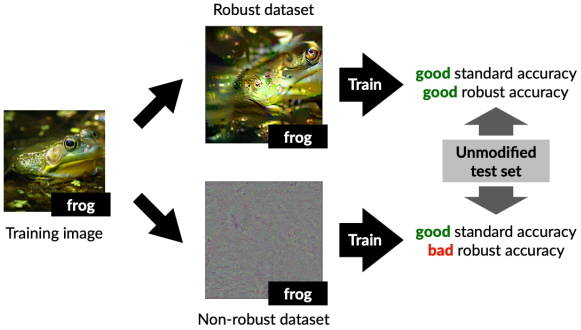


Figure 1: Example of the first experiment

Contributions (II)

- A non-robust version for standard classification. We are also able to construct a training dataset for which the inputs are nearly identical to the originals, but all appear incorrectly labeled. In fact, the inputs in the new training set are associated to their labels only through small adversarial perturbations (and hence utilize only non-robust features). Despite the lack of any predictive human-visible information, training on this dataset yields good accuracy on the original, unmodified test set. This demonstrates that adversarial perturbations can arise from flipping features in the data that are useful for classification of correct inputs (hence not being purely aberrations).

Contributions (II)

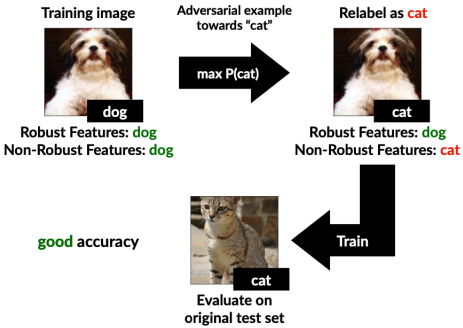


Figure 2: examples of the (appearing to be) mislabled dataset

Definitions

- set of all features $\mathcal{F} = \{f : X \to \mathbb{R}\}$
- ρ-useful feature $\mathbb{E}_{x,y\sim\mathcal{D}}[y.f(x)] \geq 0$
- γ-robustly useful features $\mathbb{E}_{x,y\sim\mathcal{D}}[\inf_{\delta\in\Delta(x)} y.f(x+\delta)] \geq \gamma$
- useful, non-robust features

Classification

- $(\mathcal{F}, \boldsymbol{\omega}, b)$ is given
- Classification $C(x) = \text{sgn}(b + \sum_{f \in \mathcal{F}} \omega_f . f(x))$
- Standard Training $\mathbb{E}_{x,y \sim \mathcal{D}}[\mathcal{L}_\theta(x, y)] = -\mathbb{E}_{x,y \sim \mathcal{D}}[y.(b + \sum_{f \in \mathcal{F}} \omega_f . f(x))]$
- Robust Training $\mathbb{E}_{x,y \sim \mathcal{D}}[\max_{\delta \in \Delta(x)} \mathcal{L}_\theta(x + \delta, y)]$

Idea

- Does training on robust features result in a robust model?
- For a neural network, feature map is the penultimate layer!
- We can make a new dataset that corresponds to the old dataset, but only robust features are left!
- How can we extract robust features??
- Let's use a robust model (adversarially-trained)!

Idea

- Does training on robust features result in a robust model?
- For a neural network, feature map is the penultimate layer!
- We can make a new dataset that corresponds to the old dataset, but only robust features are left!
- How can we extract robust features??
- Let's use a robust model (adversarially-trained)!

Idea

- Does training on robust features result in a robust model?
- For a neural network, feature map is the penultimate layer!
- We can make a new dataset that corresponds to the old dataset, but only robust features are left!
- How can we extract robust features??
- Let's use a robust model (adversarially-trained)!

Idea

- Does training on robust features result in a robust model?
- For a neural network, feature map is the penultimate layer!
- We can make a new dataset that corresponds to the old dataset, but only robust features are left!
- How can we extract robust features??
- Let's use a robust model (adversarially-trained)!

Idea

- Does training on robust features result in a robust model?
- For a neural network, feature map is the penultimate layer!
- We can make a new dataset that corresponds to the old dataset, but only robust features are left!
- How can we extract robust features??
- Let's use a robust model (adversarially-trained)!

Algorithm(Aim)

- given a robust model C, we aim to construct a distribution $\hat{\mathcal{D}_R}$ so that:

$$\mathbb{E}_{(x,y)\sim\hat{\mathcal{D}_R}}[f(x).y] = \begin{cases} \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}}[f(x).y] & \text{if } f \in \mathcal{F}_C \\ 0 & \text{otherwise,} \end{cases}$$

  Where $\mathcal{F}_C$ is the set of features utilized by C.
- But how can we extract features in the high-dimentional space?

Algorithm(implementation)

- $g(x)$ is the feature vector in the adversarially-trained model.
- For each x, we perform the following: $\min_{x_r} ||g(x_r) - g(x)||_2$
- What should be the starting point of the algorithm?

Algorithm(Pseudo-code)

GetRobustDataset(D)

1. $C_R \leftarrow$ AdversarialTraining(D)
   $g_R \leftarrow$ mapping learned by C_R from the input to the representation layer
2. $D_R \leftarrow \{\}$
3. For $(x, y) \in D$
   $x' \sim D$
   $x_R \leftarrow \arg\min_{z \in [0,1]^d} \|g_R(z) - g_R(x)\|_2$ # Solved using ℓ_2-PGD starting from x'
   $D_R \leftarrow D_R \bigcup \{(x_R, y)\}$
4. Return D_R

Figure 3: Pseudo-code for extracting robust featuers

Robust and non-Robust dataset examples

Figure 4: dataset examples

Results of Training on Robust and non-Robust Datasets

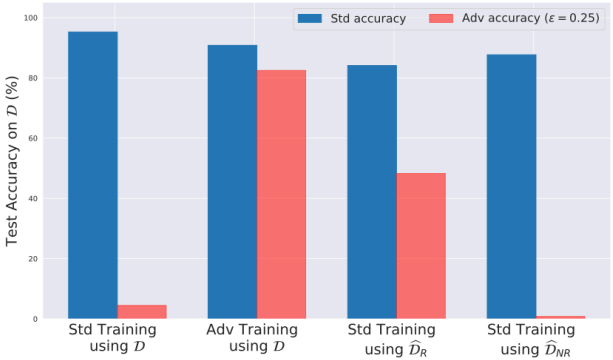


Figure 5: training resuilts

Motivation and Questions

- Are non-robust features sufficient for generalization?
- How non-robust features perform in the presence of robust features?
- Can we train a classifier solely on non-robust featurs and expect good results?

Motivation and Questions

- Are non-robust features sufficient for generalization?
- How non-robust features perform in the presence of robust features?
- Can we train a classifier solely on non-robust featurs and expect good results?

Motivation and Questions

- Are non-robust features sufficient for generalization?
- How non-robust features perform in the presence of robust features?
- Can we train a classifier solely on non-robust featurs and expect good results?

Idea

- Let's create a new dataset again.
- Now we want to set up a competition between robust and non-robust features. We don't want to delete any of them.
- So let's make each of them associated (in terms of the framework: useful) with different labels!

Idea

- Let's create a new dataset again.
- Now we want to set up a competition between robust and non-robust features. We don't want to delete any of them.
- So let's make each of them associated (in terms of the framework: useful) with different labels!

Idea

- Let's create a new dataset again.
- Now we want to set up a competition between robust and non-robust features. We don't want to delete any of them.
- So let's make each of them associated (in terms of the framework: useful) with different labels!

Algorithm(Aim)

- t is going to be the target class
- If we choose the target class randomly, we must have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{non-robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

- If we choose the target class deterministically(constant for each class), we expect to have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{non-robustly useful under } \mathcal{D} \\ < 0 & \text{if } f \text{robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

Algorithm(Aim)

- t is going to be the target class
- If we choose the target class randomly, we must have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{non-robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

- If we choose the target class deterministically(constant for each class), we expect to have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{non-robustly useful under } \mathcal{D} \\ < 0 & \text{if } f \text{robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

Algorithm(Aim)

- t is going to be the target class
- If we choose the target class randomly, we must have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{ non-robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

- If we choose the target class deterministically(constant for each class), we expect to have the following:

$$\mathbb{E}_{(x,t)\sim\hat{\mathcal{D}}_{Rand}}[f(x).t] \begin{cases} > 0 & \text{if } f \text{ non-robustly useful under } \mathcal{D} \\ < 0 & \text{if } f \text{ robustly useful under } \mathcal{D} \\ \approx 0 & \text{otherwise,} \end{cases}$$

Idea Towards Math

- Given standard (non-robust) classifier C, For each datapoint we solve the following: $x_{adv} = \underset{||x'-x|| \leq \epsilon}{argmin}\ \mathcal{L}_C(x', t)$ in which, t is the target class.
- We can use a adversarial attack to solve the optimization!
- After solving the optimization problems, we train the network using this new dataset. Keep in mind that we will use t as the target for the network. It's going to look mislabeled to humans!

Idea Towards Math

- Given standard (non-robust) classifier C, For each datapoint we solve the following: $x_{adv} = \underset{||x'-x|| \leq \epsilon}{argmin}\ \mathcal{L}_C(x', t)$ in which, t is the target class.
- We can use a adversarial attack to solve the optimization!
- After solving the optimization problems, we train the network using this new dataset. Keep in mind that we will use t as the target for the network. It's going to look mislabeled to humans!

Idea Towards Math

- Given standard (non-robust) classifier C, For each datapoint we solve the following: $x_{adv} = \underset{||x'-x|| \leq \epsilon}{argmin}\ \mathcal{L}_C(x', t)$ in which, t is the target class.
- We can use a adversarial attack to solve the optimization!
- After solving the optimization problems, we train the network using this new dataset. Keep in mind that we will use t as the target for the network. It's going to look mislabeled to humans!

Pseudo-Code

GetNonRobustDataset(D, ε)

1. $D_{NR} \leftarrow \{\}$
2. $C \leftarrow$ StandardTraining(D)
3. For $(x, y) \in D$
   $t \overset{\text{uar}}{\sim} [C]$ # or $t \leftarrow (y+1) \mod C$
   $x_{NR} \leftarrow \min_{||x'-x|| \leq \varepsilon} L_C(x', t)$ # Solved using ℓ_2 PGD
   $D_{NR} \leftarrow D_{NR} \bigcup \{(x_{NR}, t)\}$
4. Return D_{NR}

Figure 6: pseudo-code for creating the non-robust dataset

Example

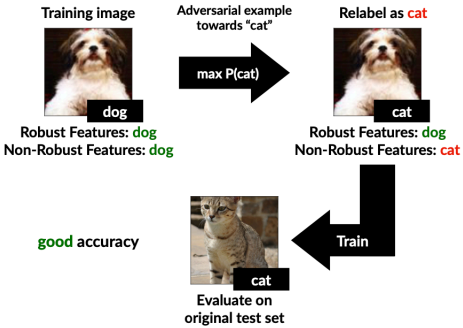


Figure 7: example of the (appearing to be) mislabled dataset

Generalization Results of Training on Mislabeled Dataset

Source Dataset	Dataset	
	CIFAR-10	ImageNet_R
$\mathcal{D}$	95.3%	96.6%
$\widehat{\mathcal{D}}_{rand}$	63.3%	87.9%
$\widehat{\mathcal{D}}_{det}$	43.7%	64.4%

Figure 8: training results

Datasets

- They used standard CIFAR10 and restricted Imagenet.

Class	Corresponding ImageNet Classes
"Dog"	151 to 268
"Cat"	281 to 285
"Frog"	30 to 32
"Turtle"	33 to 37
"Bird"	80 to 100
"Primate"	365 to 382
"Fish"	389 to 397
"Crab"	118 to 121
"Insect"	300 to 319

Figure 9: Classes used in the Restricted ImageNet model.

Models

- They used a ResNet-50 throughout the paper.
- For each setting, they tested different values for learning rate, learning rate scheduler, and batch size.

Dataset	LR	Batch Size	LR Drop	Data Aug.	Momentum	Weight Decay
$\hat{\mathcal{D}}_R$ (CIFAR)	0.1	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_R$ (Restricted ImageNet)	0.01	128	No	Yes	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_{NR}$ (CIFAR)	0.1	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_{rand}$ (CIFAR)	0.01	128	Yes	Yes	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_{rand}$ (Restricted ImageNet)	0.01	256	No	No	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_{det}$ (CIFAR)	0.1	128	Yes	No	0.9	$5 \cdot 10^{-4}$
$\hat{\mathcal{D}}_{det}$ (Restricted ImageNet)	0.05	256	No	No	0.9	$5 \cdot 10^{-4}$

Figure 10: models specifiactions

Adversarial Attack

- They used a 7-step PGD with a stepsize of $\epsilon/5$

Adversary	CIFAR-10	Restricted Imagenet
ℓ_2	0.5	3

Figure 11: adversary

Motivation

- Transferability of adversarial examples among different models has been seen so far in the litaruture.
- This hyposethis suggests that this phenomenon is a direct result of models learning the same non-robust features in the dataset!

Experiment Setup

- They crafted a non-robust dataset using a ResNet-50 (like the second experiment).
- Then, they created adversarial examples using the same model.
- Their hypothesis would suggest that architectures which learn better from this training set (in terms of performance on the standard test set) are more likely to learn similar non-robust features to the original classifier.

Experimental Proof

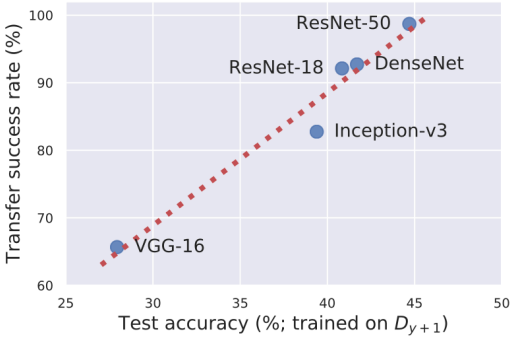


Figure 12: transfer results: test accuracy is correlated with the transfer rate!

[1] A. I. et al., "Adversarial examples are not bugs, they are features," 2019.

Thank You

Intriguing Properties of Vision Transformers

Authors: Muzammal Naseer, Kanchana Ranasinghe, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang

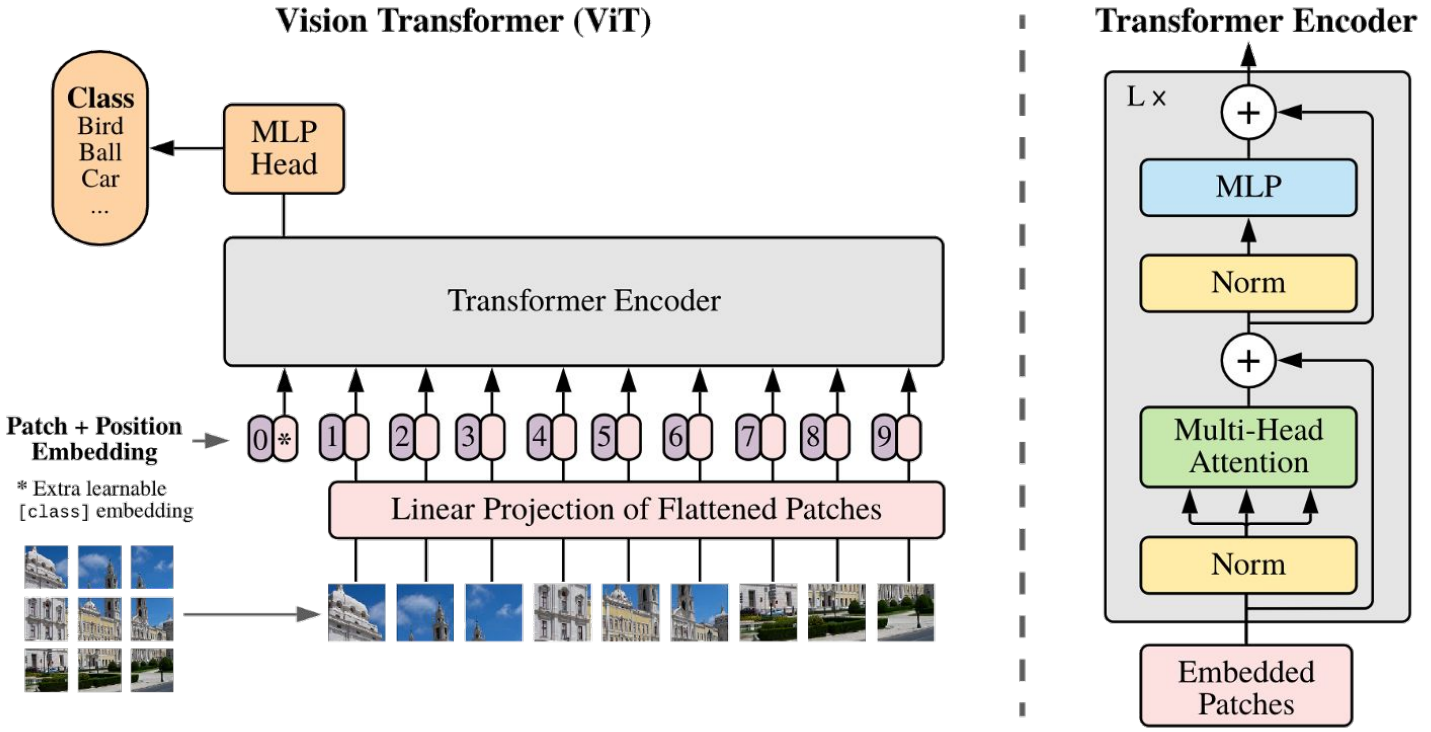
Feraidoon Mehri
Amirhossein Hadian

Goal

- Studies Vision Transformers (ViT) and compares them to CNNs
  - Robustness to
    - Occlusion
    - Natural corruptions (rain, fog, etc. effect)
    - Permutation
    - FGSM
    - PGD
    - Adversarial patch
  - Shape bias (versus texture bias)
  - Transfer learning
    - Learning simple (linear) classifiers on the model's learned (and frozen) features

Vision Transformer Review

ViT: Architecture



ViT Models

- Our in-depth analysis is based on three transformer families:
  - ViT
  - DeiT
    - DeiT-S (22-Million params) is comparable to ResNet50 (23-Million params) in size
  - T2T

Model Variants

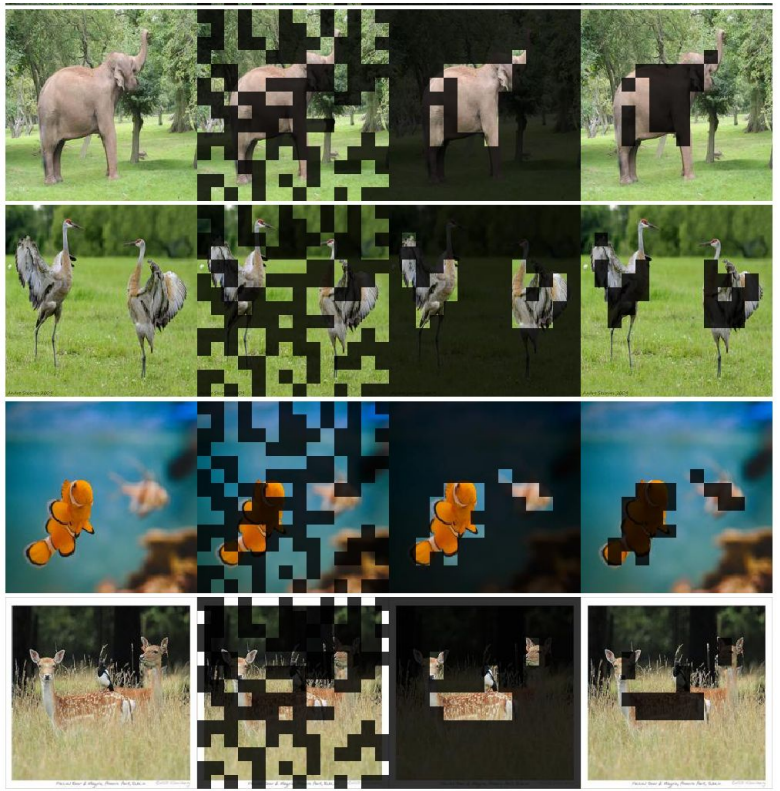
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Model	ViT model	embedding dimension	#heads	#layers	#params	training resolution	throughput (im/sec)
DeiT-Ti	N/A	192	3	12	5M	224	2536
DeiT-S	N/A	384	6	12	22M	224	940
DeiT-B	ViT-B	768	12	12	86M	224	292

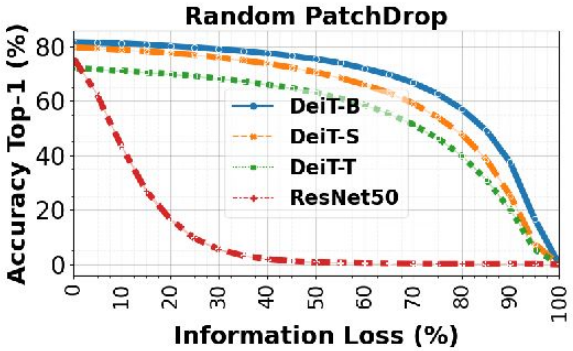
Occlusion

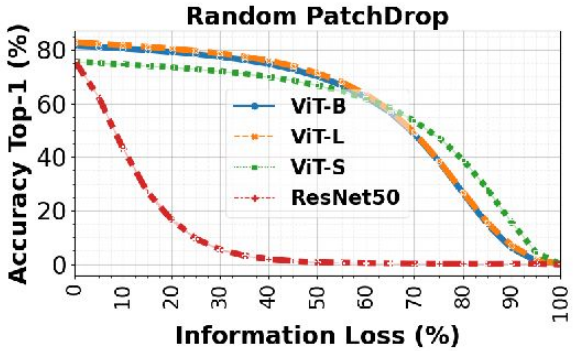
Different Occlusion Strategies

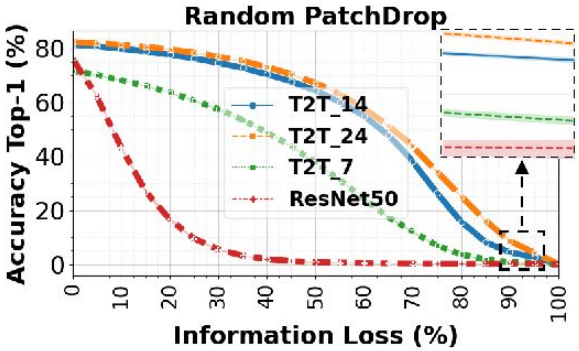
- Random PatchDrop
- Non-salient (background) PatchDrop
- Salient (foreground) PatchDrop
  - Not important for us to know how the salient parts are determined
  - The Self-supervised ViT model, DINO, can be used



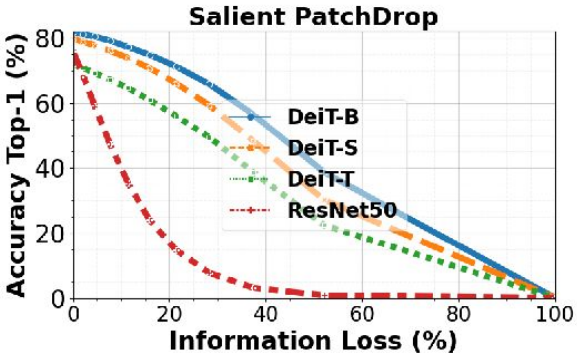
Performance Against Random PatchDrop

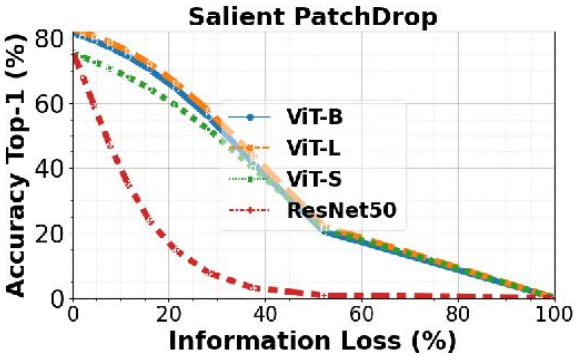


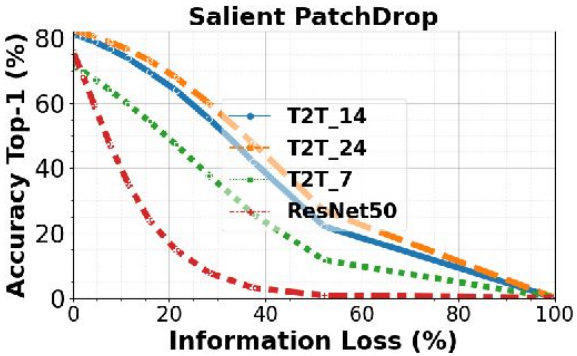




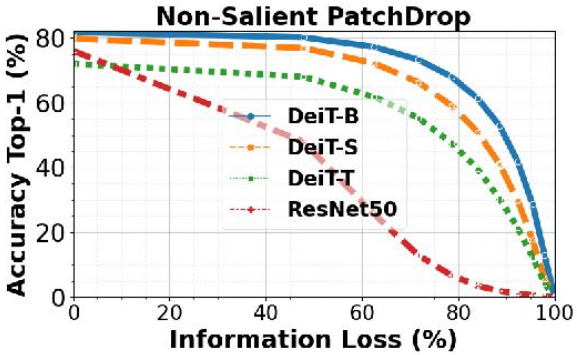
Performance Against Salient PatchDrop

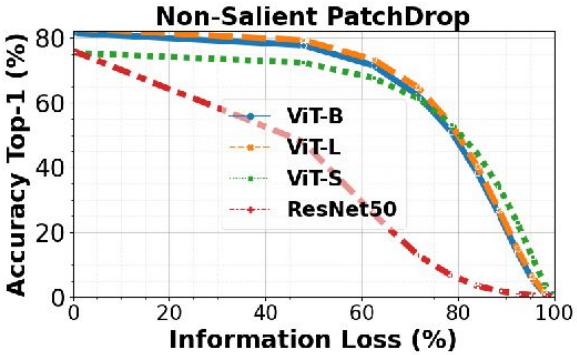


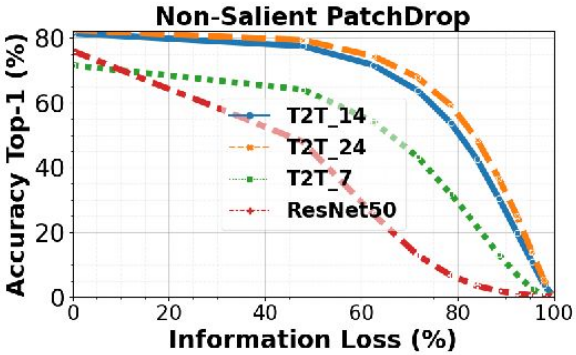




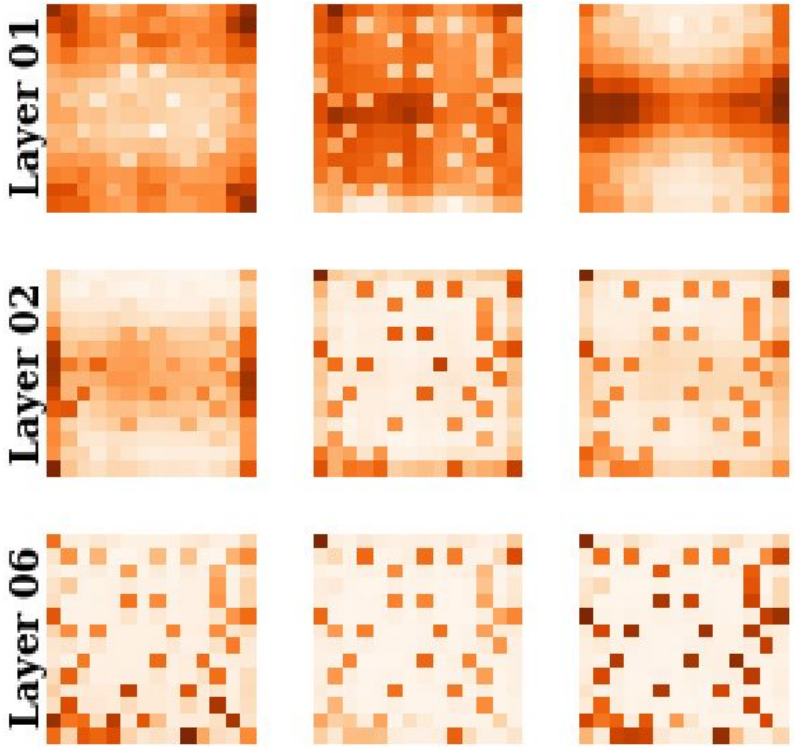
Performance Against Non-Salient PatchDrop



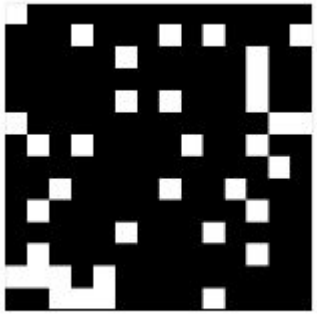




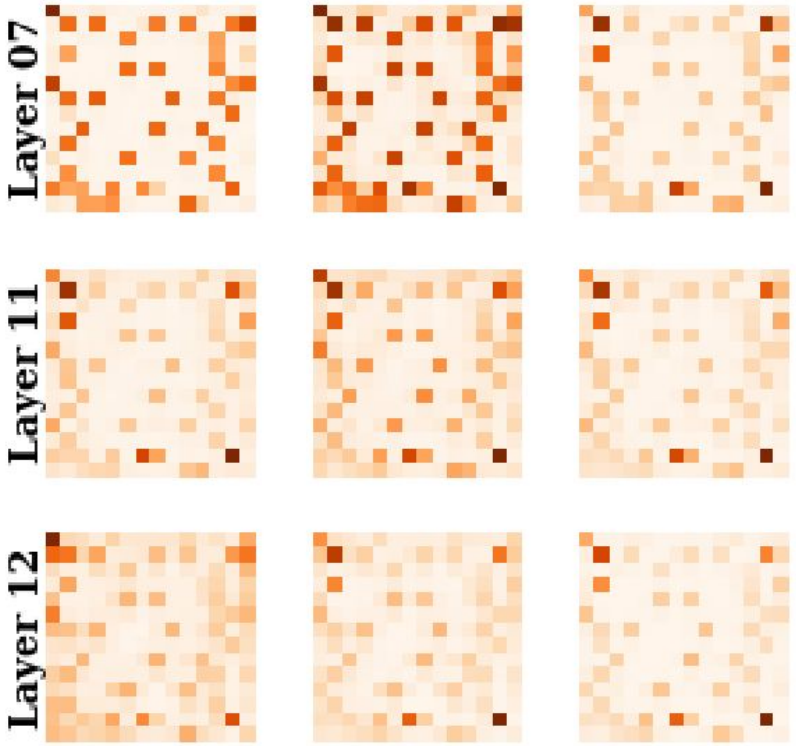
Attention Maps Relevant to Each Head

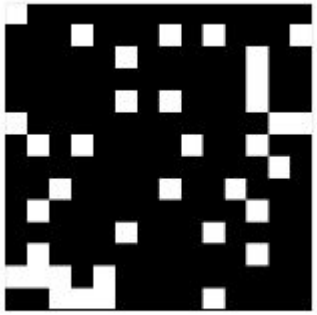


- ImageNet pre-trained DeiT-B model
- Attention maps are averaged over the entire ImageNet val. Set
- Same mask is used for all samples (mask is shown below)



Attention Maps Relevant to Each Head





Later Layers Attend to Non-Occluded Regions

Shape Bias

CNNs Recognize Textures, Not Shapes

- CNNs mainly exploit texture to make a decision and give less importance to global shape
  - In stark contrast to human behavioural evidence

CNNs Recognize Textures, Not Shapes

(a) Texture image

81.4%	**Indian elephant**
10.3%	indri
8.2%	black swan

(b) Content image

71.1%	**tabby cat**
17.3%	grey fox
3.3%	Siamese cat

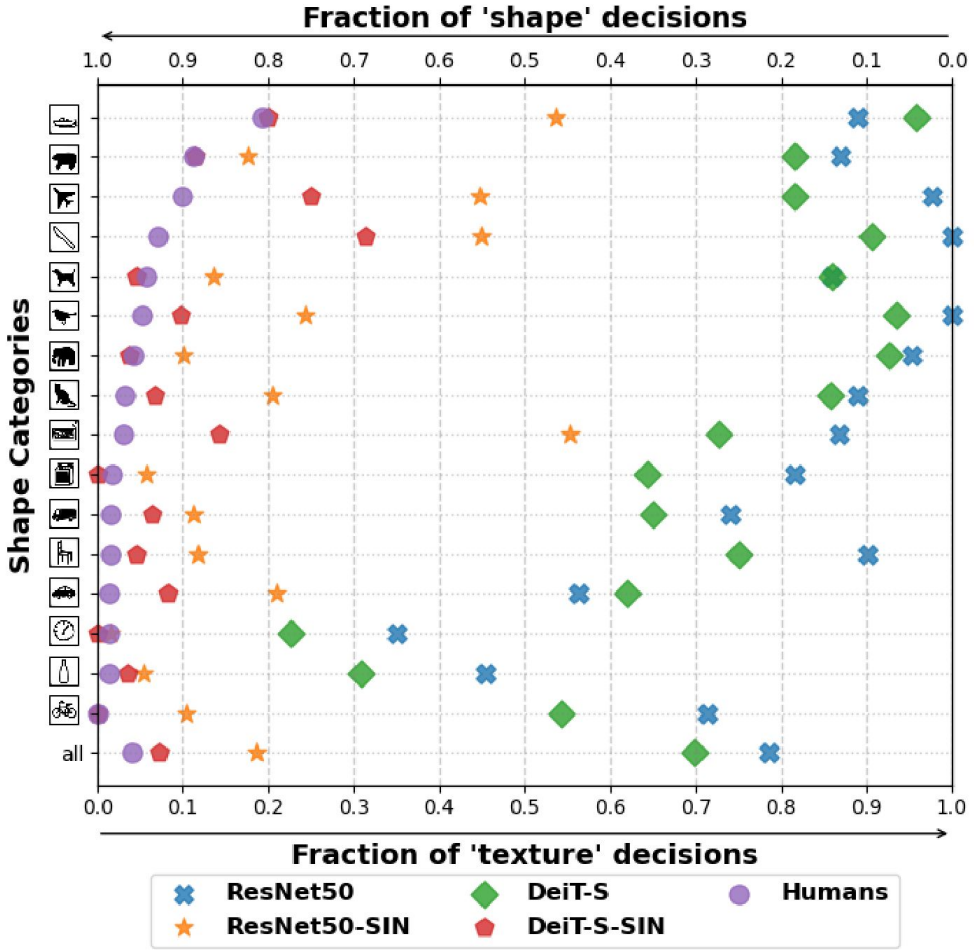
(c) Texture-shape cue conflict

63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

Figure 1: Classification of a standard ResNet-50 of **(a)** a texture image (elephant skin: only texture cues); **(b)** a normal image of a cat (with both shape and texture cues), and **(c)** an image with a texture-shape cue conflict, generated by style transfer between the first two images.

SIN: Stylized-ImageNet

- Geirhos et al. demonstrate that ResNet-50 is able to learn a shape-based representation instead when trained on "Stylized-ImageNet", a stylized version of ImageNet.


Fraction of 'shape' decisions
1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0
Shape Categories
all
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Fraction of 'texture' decisions
ResNet50
ResNet50-SIN
DeiT-S
DeiT-S-SIN
Humans

Shape Bias: ViTs Vs. CNNs

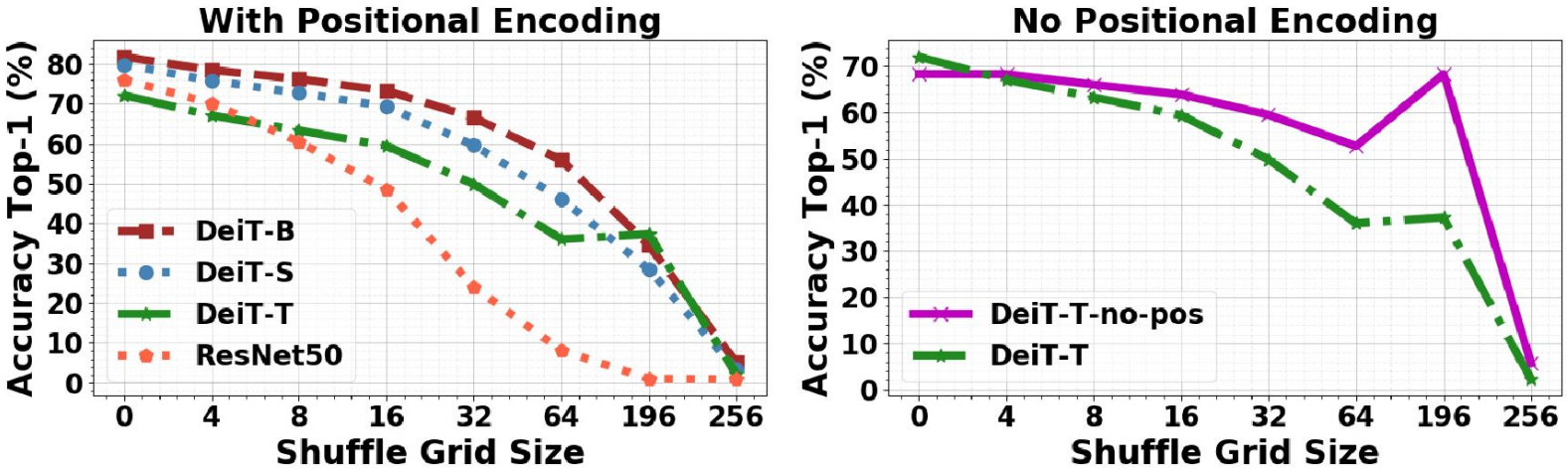
- Overall, ViTs perform better than CNN
- The shape bias increases significantly when trained on stylized ImageNet (SIN)

Permutation

Shuffling Strategies

Original | 2 x 2 Grid | 4 x 4 Grid | 8 x 8 Grid | 14 x 14 Grid

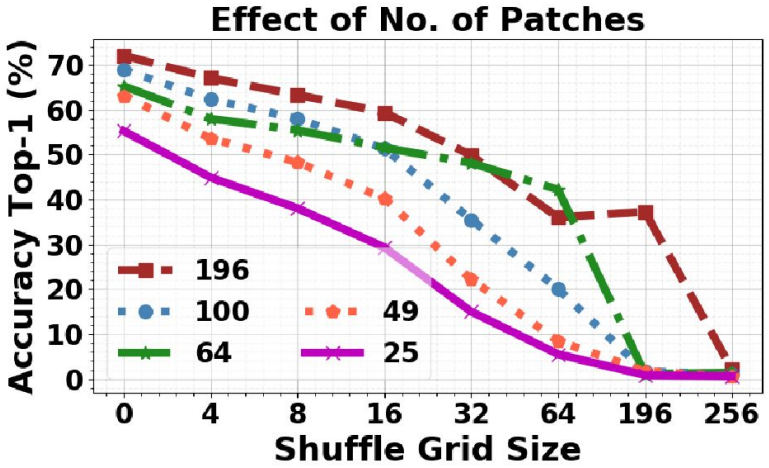
Performance Against Shuffling



- Models trained on 196 image patches
  - The model trained without positional encodings is invariant to shuffling with a grid size of 196

Effect of Patch Size on Permutation Robustness

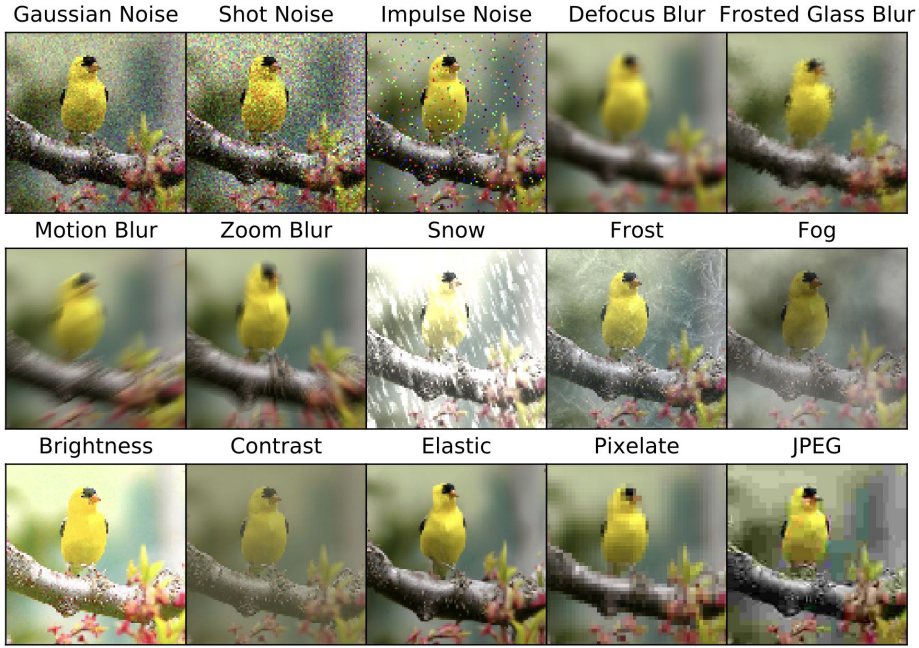
- DeiT-T trained on different numbers of image patches



Natural Corruptions

Natural Corruptions

- Synthetic common corruptions (e.g., rain, fog, snow and noise)



Natural Corruptions

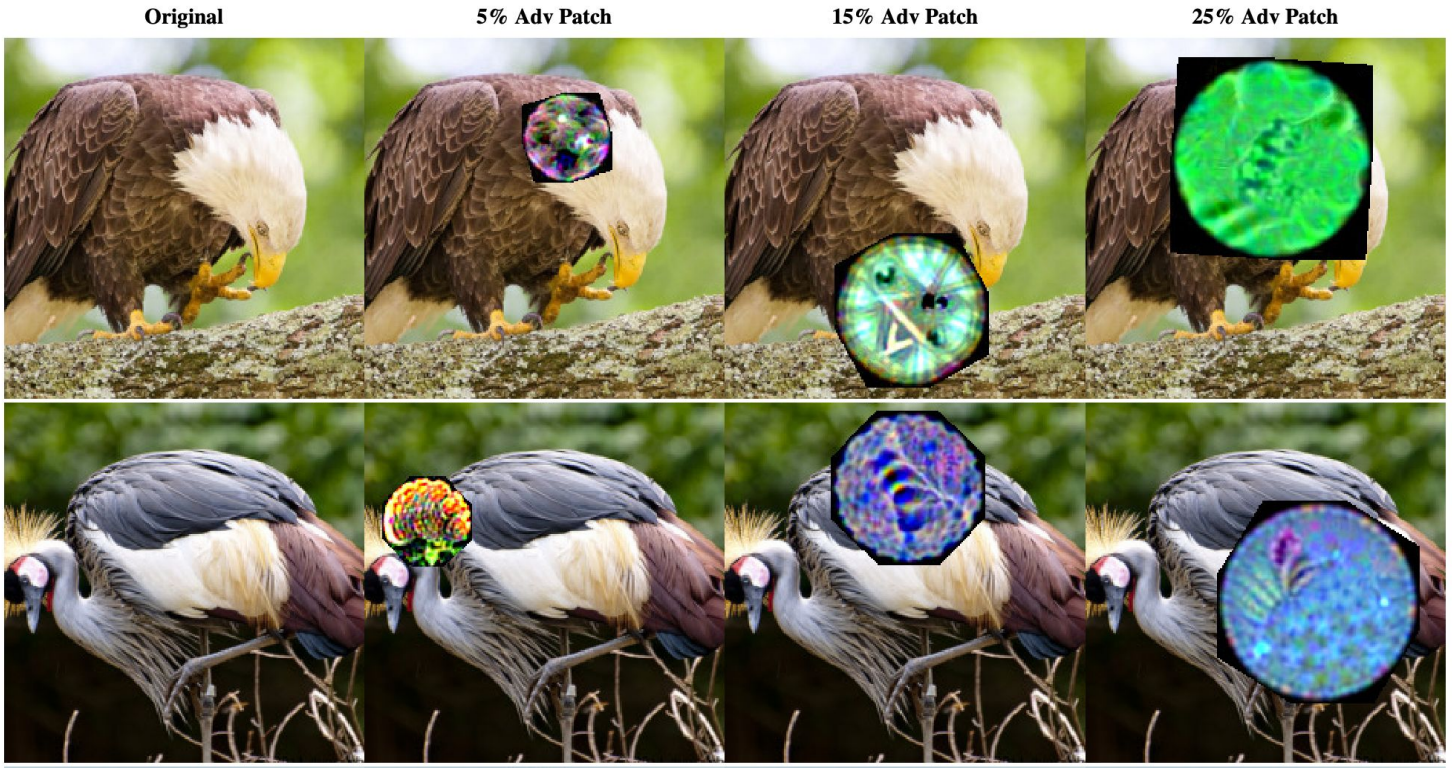
- A ViT with similar parameters as CNN (e.g., DeiT-S) is more robust to image corruptions than ResNet50 trained with augmentations (Augmix)

Trained with Augmentations						Trained without Augmentation			
DeiT-B	DeiT-S	DeiT-T	T2T-24	TnT-S	Augmix	ResNet50	ResNet50-SIN	DeiT-T-SIN	DeiT-S-SIN
48.5	54.6	71.1	49.1	53.1	65.3	76.7	77.3	94.4	84.0

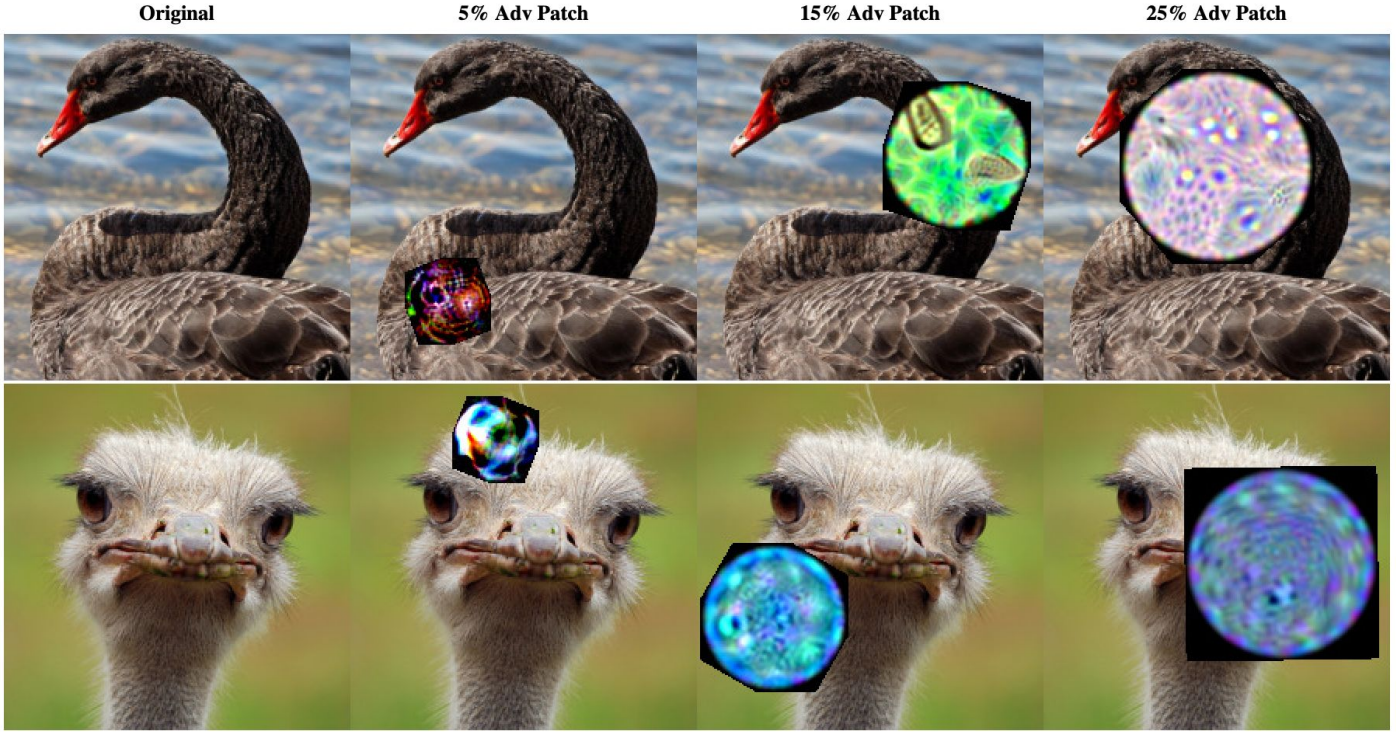
Table 4: mean Corruption Error (mCE) across common corruptions [13] (lower the better). While ViTs have better robustness compared to CNNs, training to achieve a higher shape-bias makes both CNNs and ViTs more vulnerable to natural distribution shifts. All models trained with augmentations (ViT or CNN) have lower mCE in comparison to models trained without augmentations on ImageNet or SIN.

Adversarial Attacks

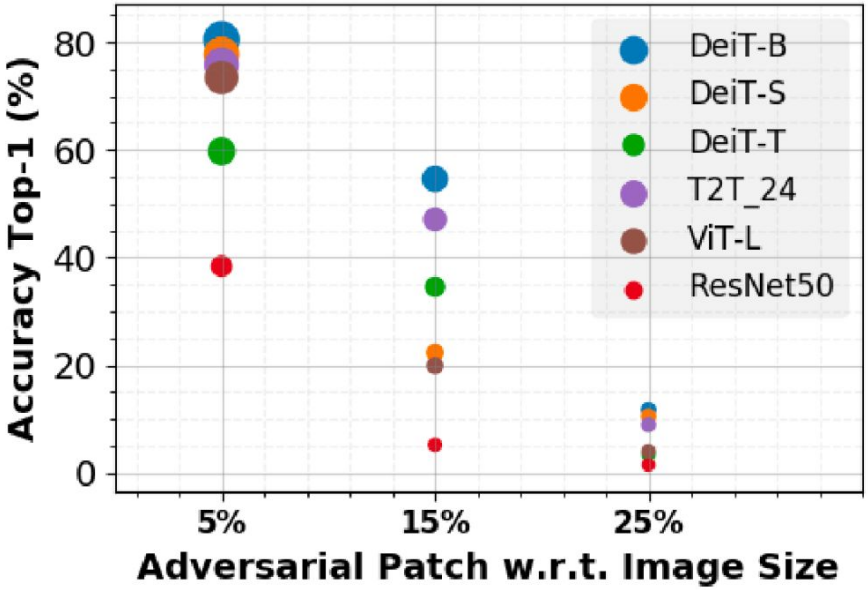
Adversarial Patches (Universal and Untargeted) Optimized to Fool DeiT-T, DeiT-B

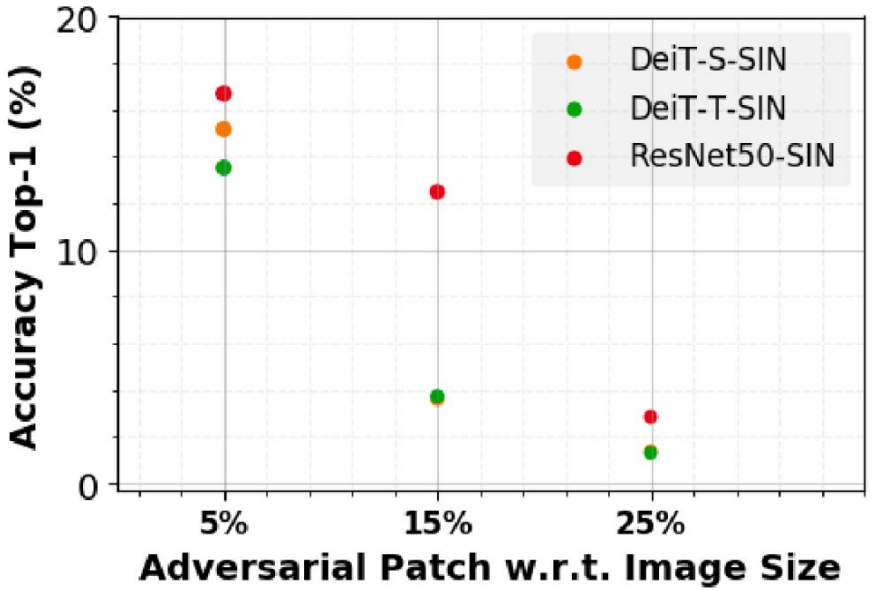


Adversarial Patches Optimized to Fool DeiT-S, DeiT-S-SIN



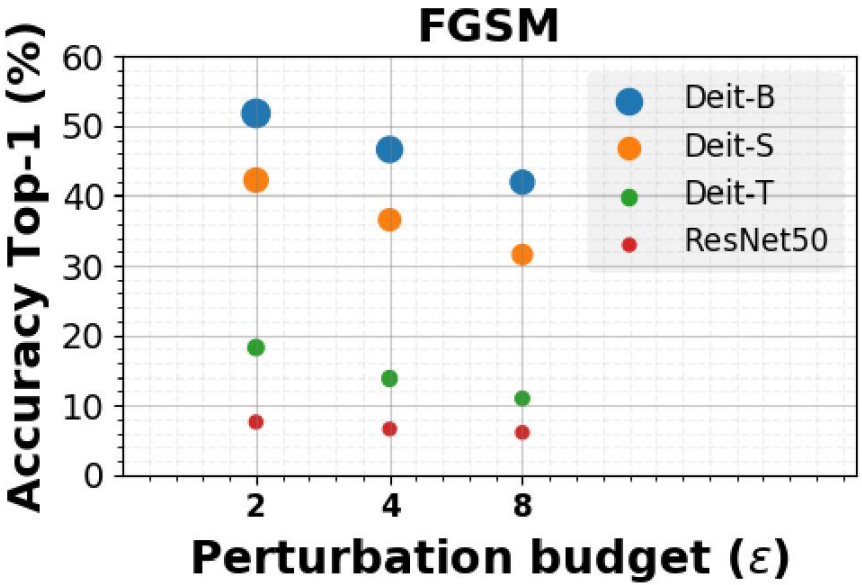
Adversarial Patch

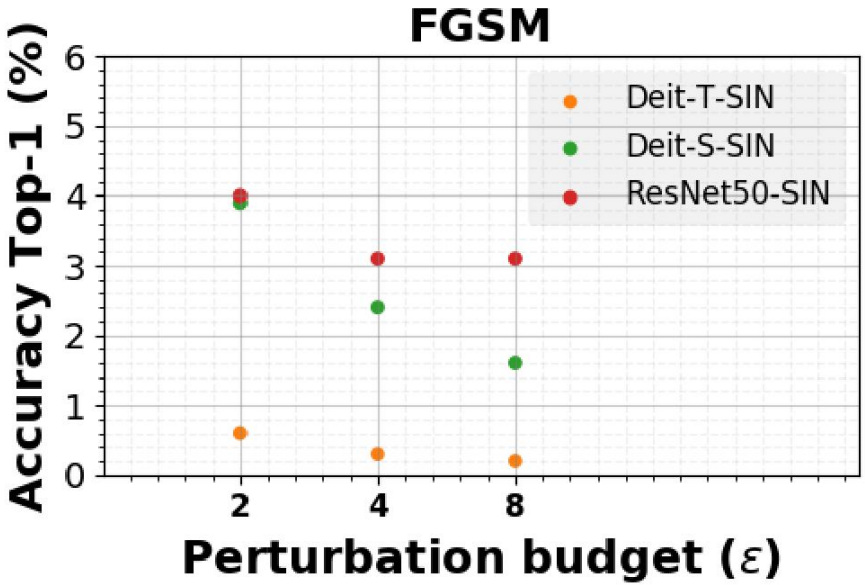




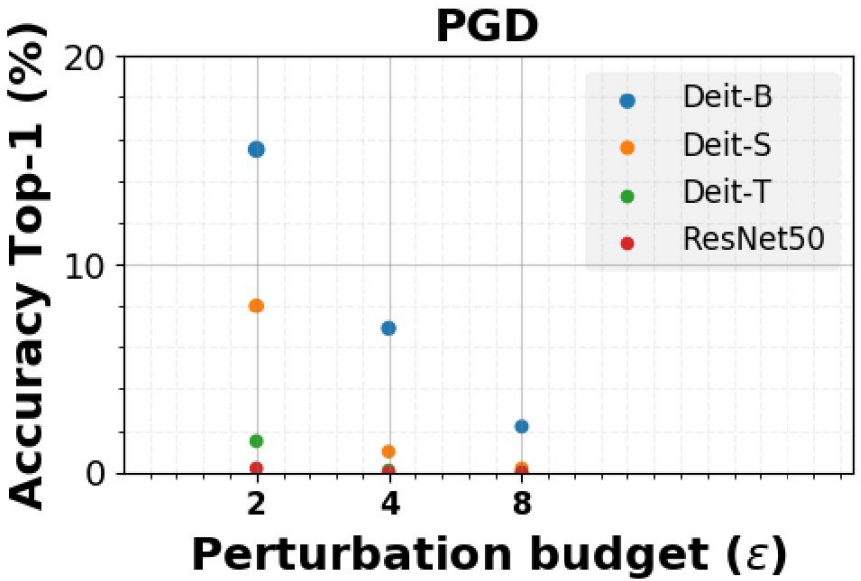
- Models trained on ImageNet are more robust than the ones trained on SIN.

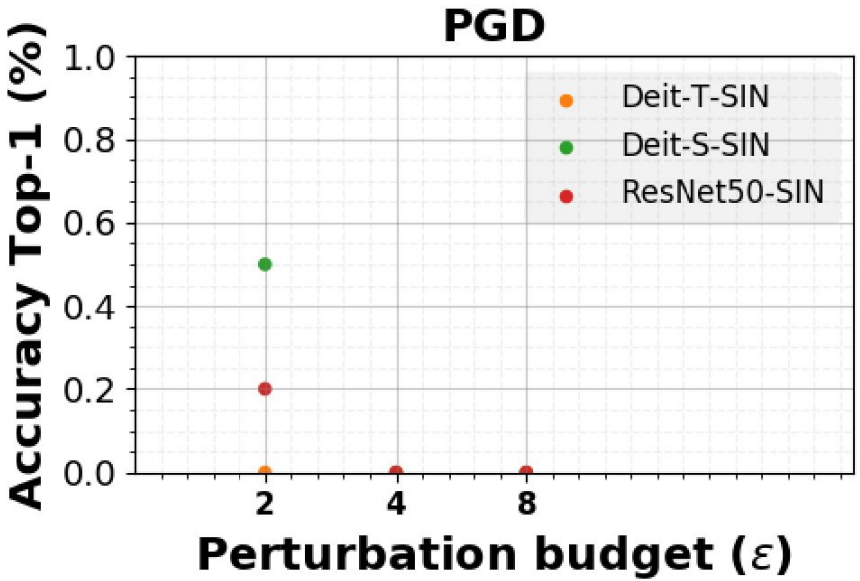
FGSM (Single-Step)





PGD (5-Step)

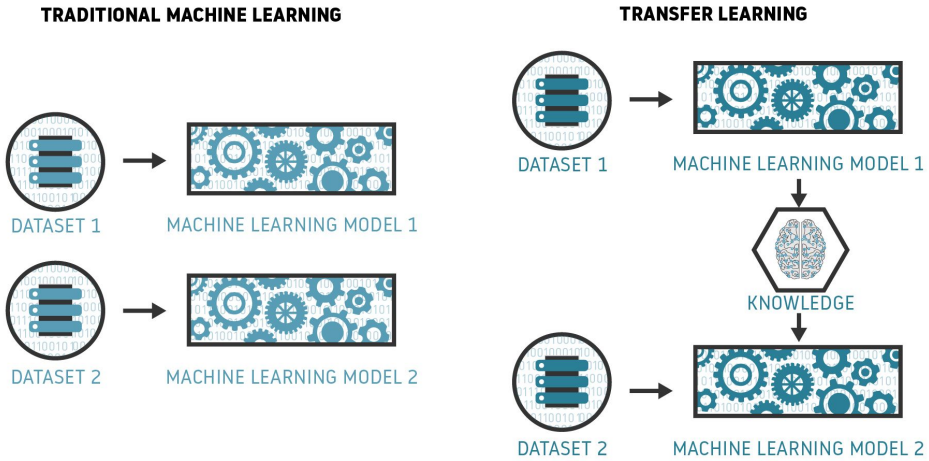

PGD
Accuracy Top-1 (%)
20
10
0
2
4
8
Perturbation budget (ε)
Deit-B
Deit-S
Deit-T
ResNet50


PGD
Accuracy Top-1 (%)
1.0
0.8
0.6
0.4
0.2
0.0
2
4
8
Perturbation budget (ε)
Deit-T-SIN
Deit-S-SIN
ResNet50-SIN

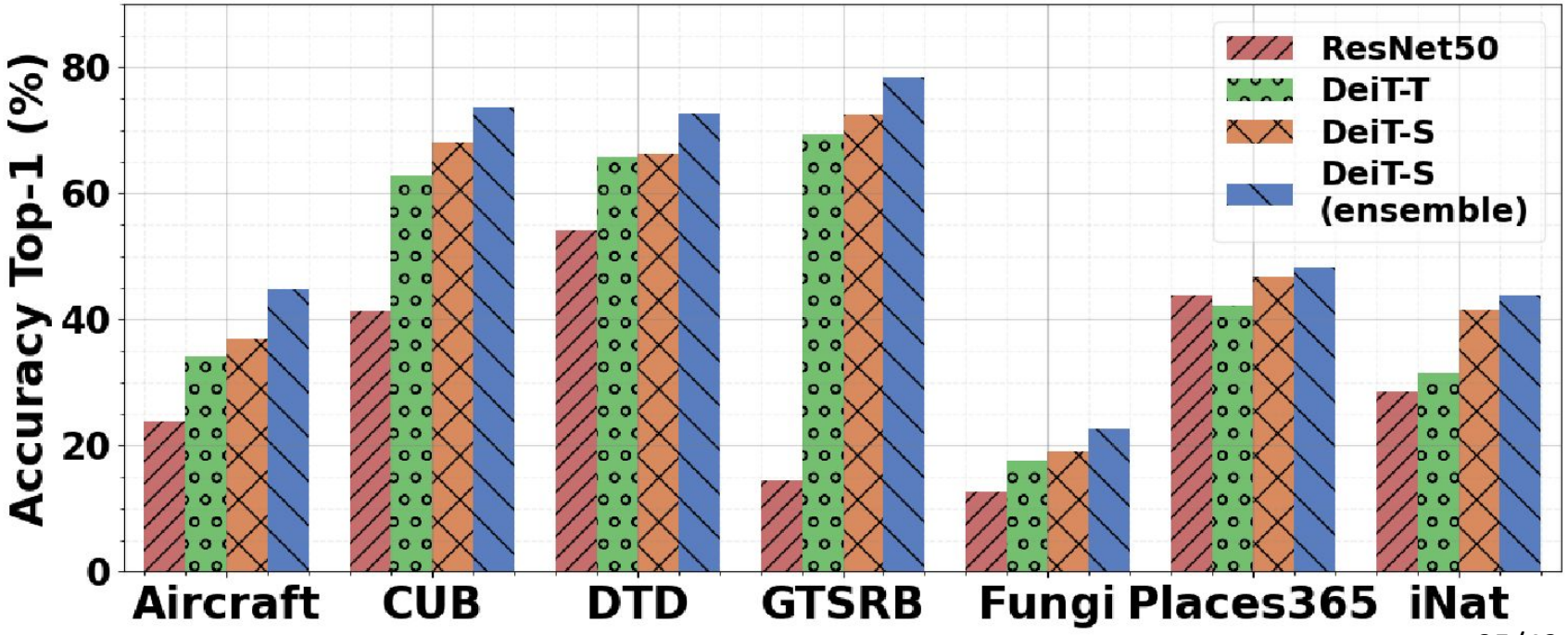
Transfer Learning

Transfer Learning

- We train a linear classifier on top of the extracted features over the train split of each dataset, and evaluate the performance on their respective test splits.



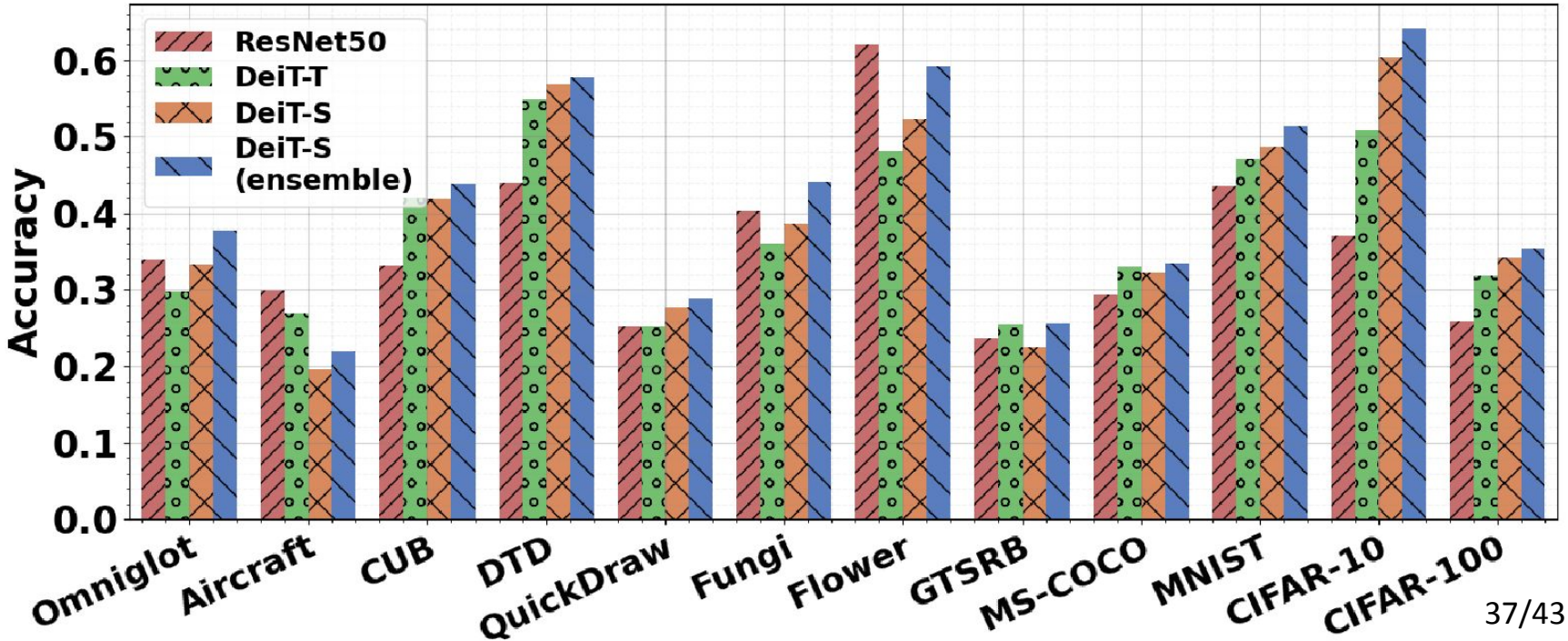
ViT Features Transfer Better Than CNNs



Few-Shot Transfer Learning (FSL)

- We use a network pre-trained for classification on ImageNet dataset to extract features.
- For each downstream dataset, under the FSL setting, a support set of labelled images is available for every test query.
- We use the extracted features to learn a linear classifier over the support set for each query.
    - This evaluation involves a varying number of shots specific for each downstream dataset.

ViTs Are Better at Few-Shot Learning



Limitations

Limitations

- Transfer learning via fine-tuning is known to be better than single-layer classifier learning on frozen features
  - Big Transfer (BiT) CNNs not tested
- Does not test the bigger models
  - Biggest model tested is ViT-Large
- Most CNN experiments only done on ResNets
- Metaformer architectures such as Google's MLP-Mixer not tested

Limitations

- Vision Transformers not systematically tested
  - Some experiments only test a single variant
- CNN/Transformer hybrids not tested
- Contrastive models not tested
  - CLIP
- Does not evaluate the robustness/dataset size trade-off

Conclusion

Conclusion

- ViTs use self-attention
  - Global, dynamic receptive field
- CNNs learn local features
- ViTs are more robust to
  - distributional shifts
  - patch permutations
  - adversarial patches
  - sample specific adversarial attacks
  - common corruptions
- ViT’s learned representations transfer better to downstream tasks

Thank you for your attention

Words Can Lie

A Presentation on
Audio Adversarial Examples: Targeted Attacks on Speech-to-Text

Mahshid Dehghani and Mehraneh Najafi

Spring 2023

Can we construct targeted adversarial example for automatic speech recognition?

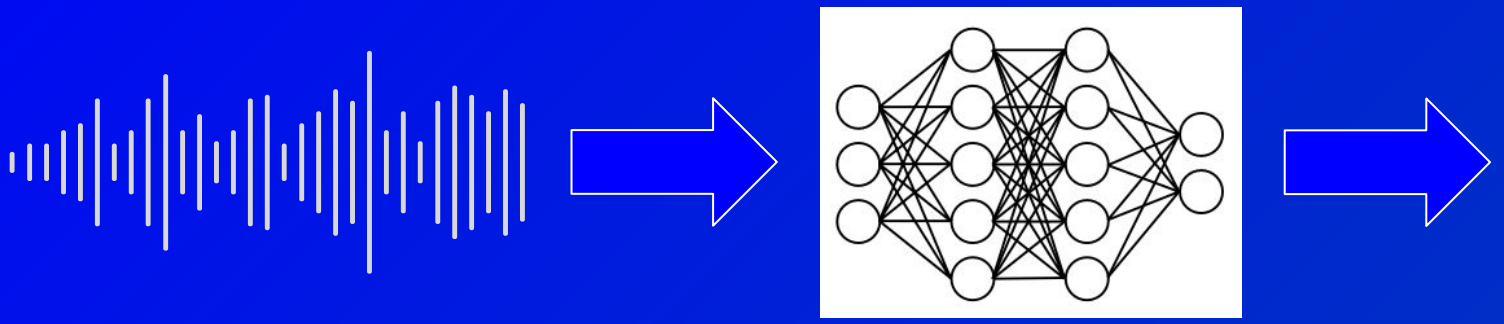
can we make a NN recognize this audio as any target transcription?
(e.g., “okey google, browse to evil.com”)

“Hey Siri”	“Hey Cortana”	“Alexa”	“OK Google”	“Hi Bixby”
				
2011	2014	2014	2016	2017

Result:

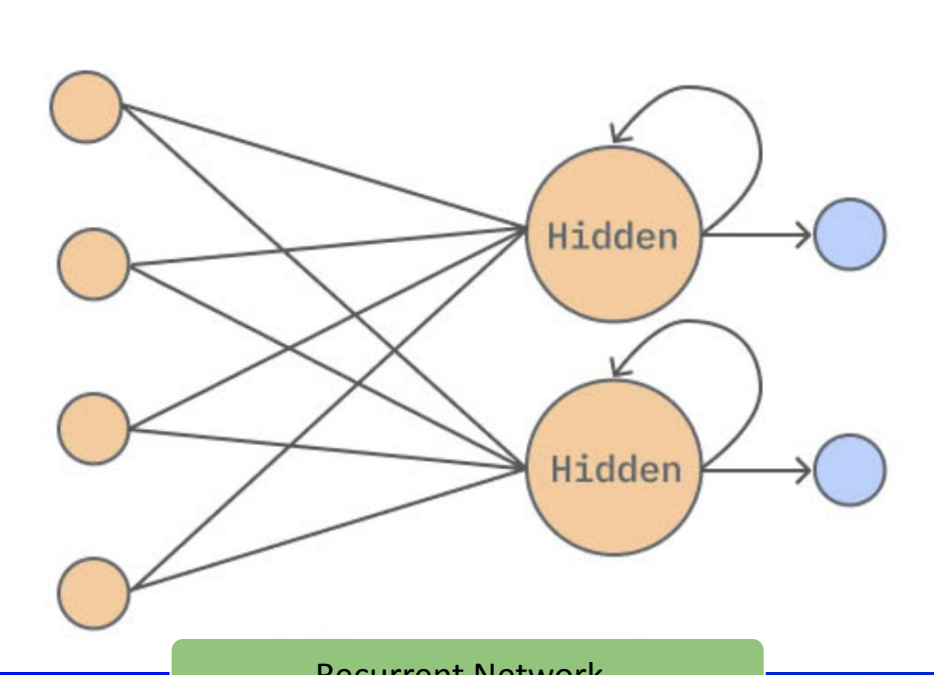
- Given any audio waveform, it can produce another that is over 99.9% similar, but transcribes as any phrase we choose.
- White-box iterative optimization-based attack to Mozilla's implementation DeepSpeech end-to-end
- 100% success rate

Neural Network for Automatic Speech Recognition

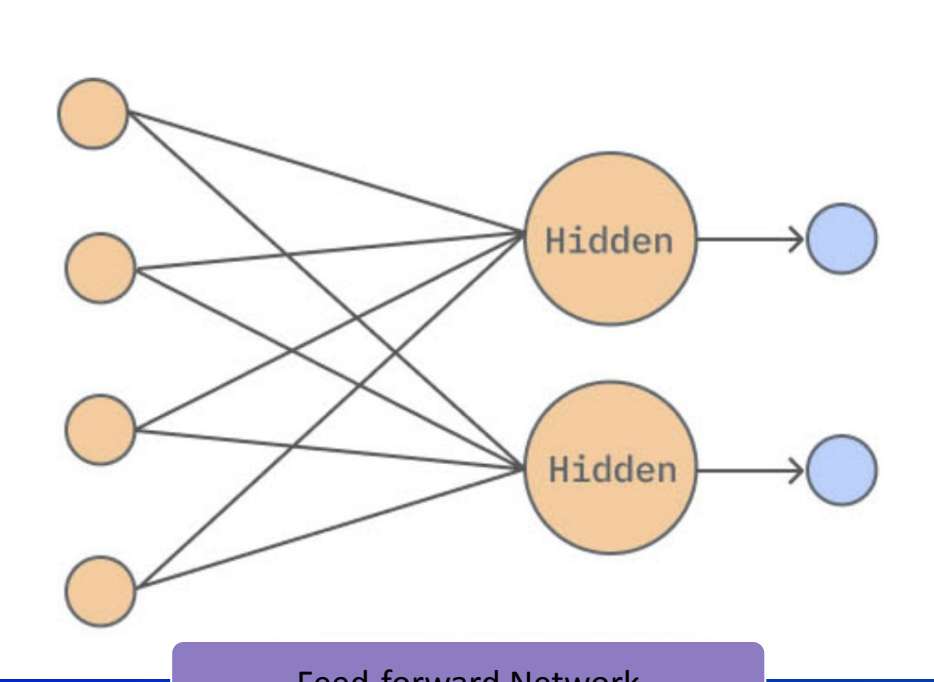


"it was the best of times, it was the worst of times"

Recurrent Neural Network

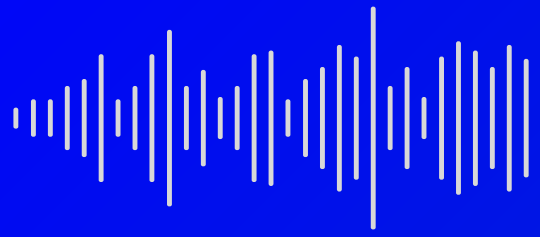


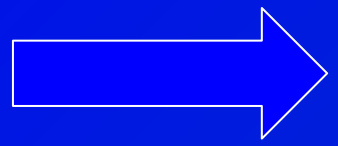
Recurrent Network



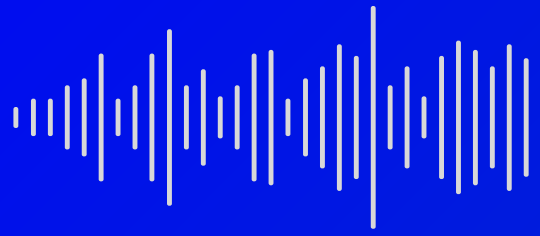
Feed-forward Network

Training Data

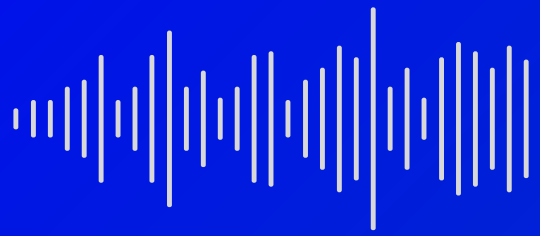




"pairs of audio and text"



"of variable length"



"with no alignment"

New Loss Function:

CTC Loss

When to use CTC?

- Many-to-many sequence prediction
- Labelling order matters, but not a one-to-one correspondence between outputs and labels

Threat Model

- Construct $x' = x + \delta$ that x and x′ sound similar x but $C(x') = y$
- Success: only if the output of the network matches exactly the target phrase (i.e., contains no misspellings or extra characters).
- Setting: white-box

Distortion Metric

- Magnitude of perturbation (in dB) relative to source audio

$$dB_x(\delta) = dB(\delta) - dB(x)$$

- Decibels (dB): a logarithmic scale that measures the relative loudness of an audio sample:

$$dB(x) = \max_i(20.\log_{10}(x_i))$$

Optimization

$$\text{minimize } dB_x(\delta)$$
$$\text{such that } C(x+\delta) = t$$
$$x + \delta \in [-M, M]$$

$$\text{minimize } dB_x(\delta) + c \cdot \ell(x+\delta, t)$$

$$\text{minimize } |\delta|_2^2 + c \cdot \ell(x+\delta, t)$$
$$\text{such that } dB_x(\delta) \leq \tau$$

Evaluation

- 100% success rate (mean perturbation of -31dB)
- The longer the phrase, the more difficult it is to target
- The longer the initial source phrase is, the easier it is to make it target a transcription
- A lot of compute time on commodity hardware for one adversarial example. Reduced to a few minutes by taking advantage of GPU's parallel nature.

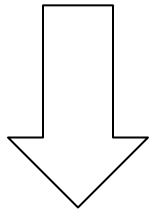
Connectionist Temporal Classification (CTC)

- X: input domain — a single frame of input
- Y: range — the characters a-z, space, and the special ε token
- neural network takes a sequence of N frames x ∈ X and returns a probability distribution over the output domain for each frame

$$f : \mathcal{X}^N \to [0,1]^{N \cdot |\mathcal{Y}|}$$

Connectionist Temporal Classification (CTC)

hello



hhhhεεεeeeεεlllllεεlloooo

Connectionist Temporal Classification (CTC)

$$\Pr(\boldsymbol{p}|\boldsymbol{y}) = \sum_{\pi \in \Pi(\boldsymbol{p},\boldsymbol{y})} \Pr(\pi|\boldsymbol{y}) = \sum_{\pi \in \Pi(\boldsymbol{p},\boldsymbol{y})} \prod_{i} \boldsymbol{y}^{i}_{\pi^i}$$

$$\textbf{CTC-Loss}(f(\boldsymbol{x}), \boldsymbol{p}) = -\log \Pr(\boldsymbol{p}|f(\boldsymbol{x})).$$

Exponential Search Space

- Greedy Decoding searches for the most likely alignment (which is easy to find) and then reduces this alignment to obtain the transcribed phrase:

$$C_{\text{greedy}}(\boldsymbol{x}) = \text{reduce}(\arg\max_{\pi} \Pr(\pi|f(\boldsymbol{x})))$$

- *Beam Search Decoding* simultaneously evaluates the likelihood of multiple alignments π and then chooses the most likely phrase p under these alignments.

The problem with CTC-loss

- Choice of loss function impacts the final distortion of adversarial example by a factor of 3 or more. The same holds in the audio domain, but to a lesser extent.
- CTC loss is highly useful for training, but a carefully designed loss function can lead to lower-distortion adversarial examples.
- In CTC, an optimizer will make every aspect of the transcribed phrase more similar to the target phrase.
  - Assume target phrase “ABCD” and we’re decoding to “ABCX”; CTC will cause “A” to be more “A”-like without the need to do so.
  - Solution:

$$l(y,t) = \max(y_t - \max_{t' \neq t} y_{t'}, 0)$$

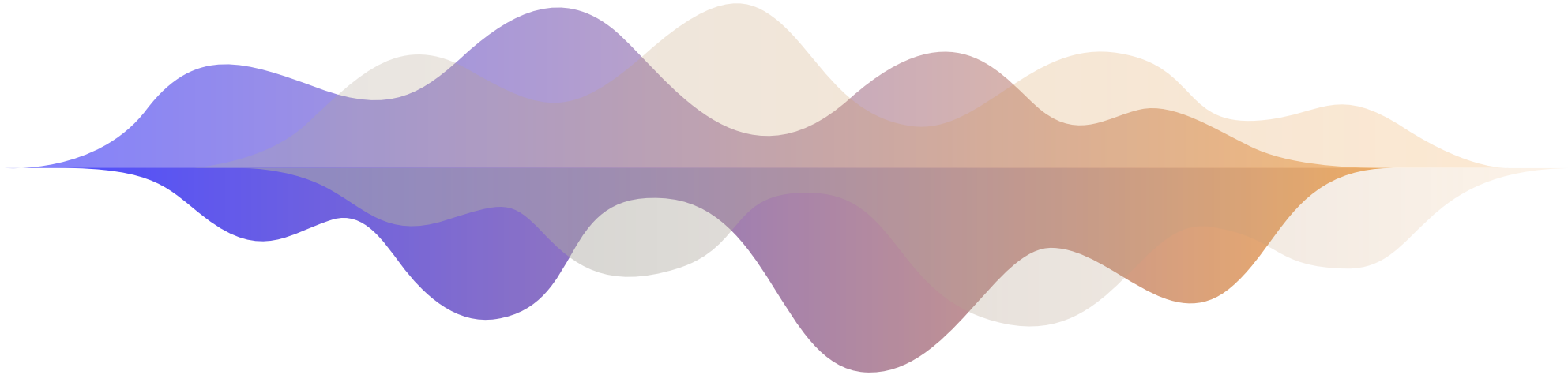
Improved Loss Function

- minimize $|\delta|_2^2 + \Sigma_i c_i . L_i(x+\delta, \pi_i)$ s.t $dB_x(\delta) < \tau$
- $L_i(x, \pi_i) = l(f(x)^i, \pi_i)$
- The constant *c* determines the relative importance of being close to the original symbol vs. being adversarial.
- The alignment loss encourages the adversarial example to have a different alignment than the original signal.
- The distance loss encourages the adversarial example to be perceptually similar to the original audio signal.
- We cannot try all alignments π, since it's computationally prohibitive.

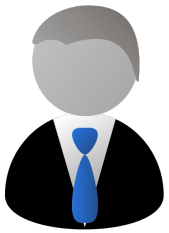
Then, what can we do?

- Two-step attack:
  - Find an initial adversarial example using CTC loss.
  - Hold the alignment π fixed and generate a less-distorted adversarial example x' targeting alignment π using the improved loss function.

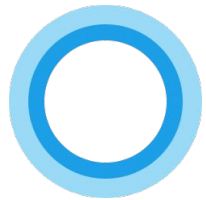
Figure 2. Original waveform (blue, thick line) with adversarial waveform (orange, thin line) overlaid; it is nearly impossible to notice a difference. The audio waveform was chosen randomly from the attacks generated and is 500 samples long.



“ Without the the dataset the article is useless.



“ Okay google browse to evil dot com.



Audio Information Density

- Input waveform is converted to 50 *frames* per second of audio and DeepSpeech outputs one probability distribution of characters per frame.
- Theoretical maximum density of audio is 50 characters per seconds.
- But what can we achieve with this information?

Audio Information Density

- Input waveform is converted to 50 *frames* per second of audio and DeepSpeech outputs one probability distribution of characters per frame.
- Theoretical maximum density of audio is 50 characters per seconds.
- But what can we achieve with this information?
  - We can generate adversarial examples with short audio clips transcribing to a long textual phrase!
- The attack is effective, though it requires a mean distortion of –18dB.

Starting from Non-Speech

Starting from Non-Speech

“speech can be embedded in music”

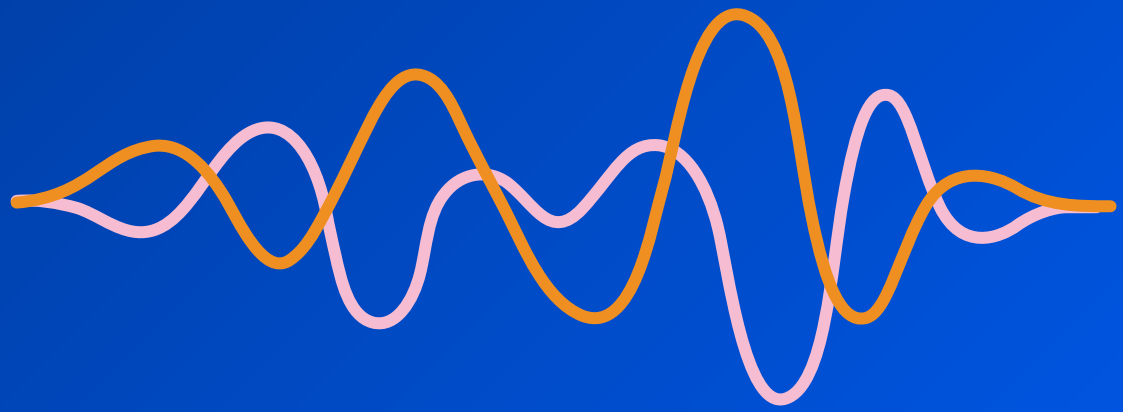
Targeting Silence

- We can also *hide* speech by adding adversarial noise.
- Performing this attack without modification (just targeting the empty phrase) is effective but we can improve it slightly by using the improved loss function we mentioned earlier.
- Targeting silence is easier than targeting a specific phrase!
  - With distortion less than -45dB below the original signal, we can turn any audio into silence.
  - Partially explains why it's easier to construct adversarial examples when starting with longer audio waveforms.

Audio Adversarial Example Properties

Single-Step methods:

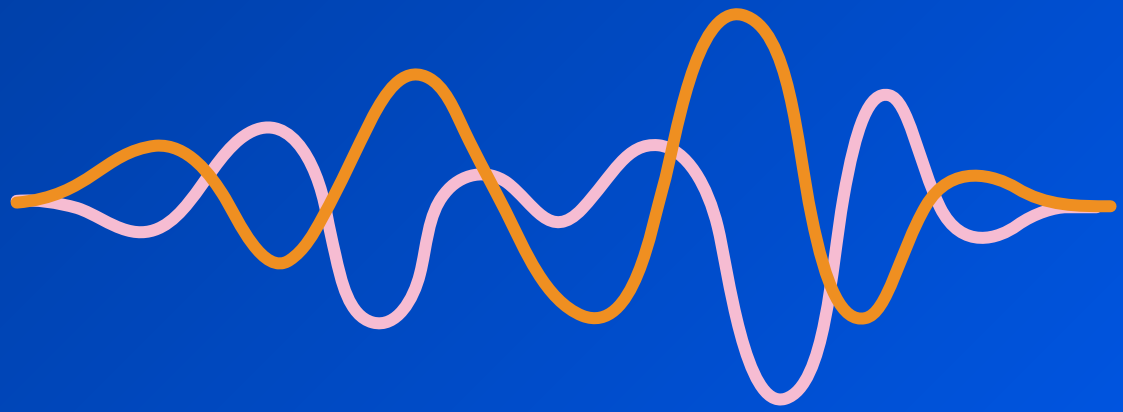
- We know how FGSM is applied on image domain. It can directly be applied to audio, changing individual samples instead of pixels.
- But how does FGSM actually perform?



Audio Adversarial Example Properties

Single-Step methods:

- We know how FGSM is applied on image domain. It can directly be applied to audio, changing individual samples instead of pixels.
- But how does FGSM actually perform?
  - Not so great!



- The inherent nonlinearity introduced in computing the MFCCs, along with the depth of many rounds of LSTMs, introduces a large degree of non-linearity in the output.
- However, FGSM can produce *untargeted* audio adversarial examples.

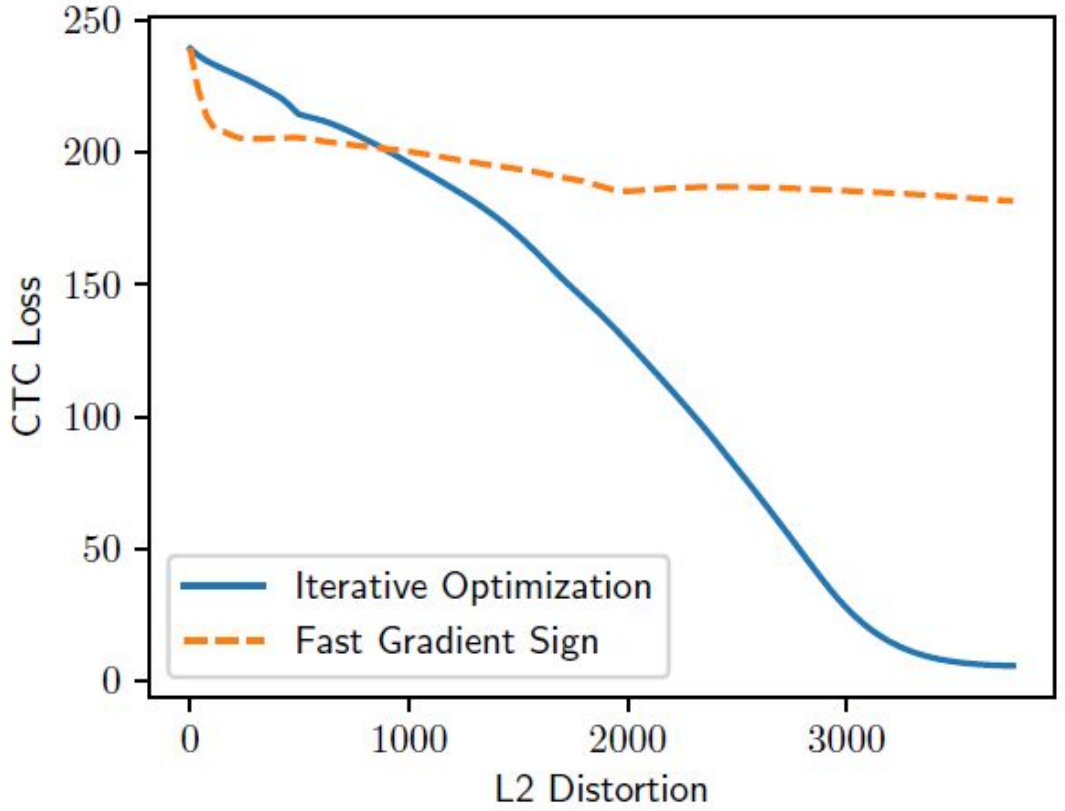


Figure 3. CTC loss when interpolating between the original audio sample and the adversarial example (blue, solid line), compared to traveling equally far in the direction suggested by the fast gradient sign method (orange, dashed line). Adversarial examples exist far enough away from the original audio sample that solely relying on the local linearity of neural networks is insufficient to construct targeted adversarial examples.

Audio Adversarial Example Properties

Robustness of Adversarial Examples:

- The adversarial examples created using our method can become non-adversarial by trivial modifications.
- We can, however, generate adversarial examples robust to pointwise noise or MP3 compression at the cost of increased perturbation. (10dB~15dB)

Open Questions

- Can these attacks be player over-the-air?
- Do universal adversarial perturbations exist?
- Are audio adversarial examples transferable?
- Which existing defenses can be applied?

Strengths and Weaknesses

Strengths:

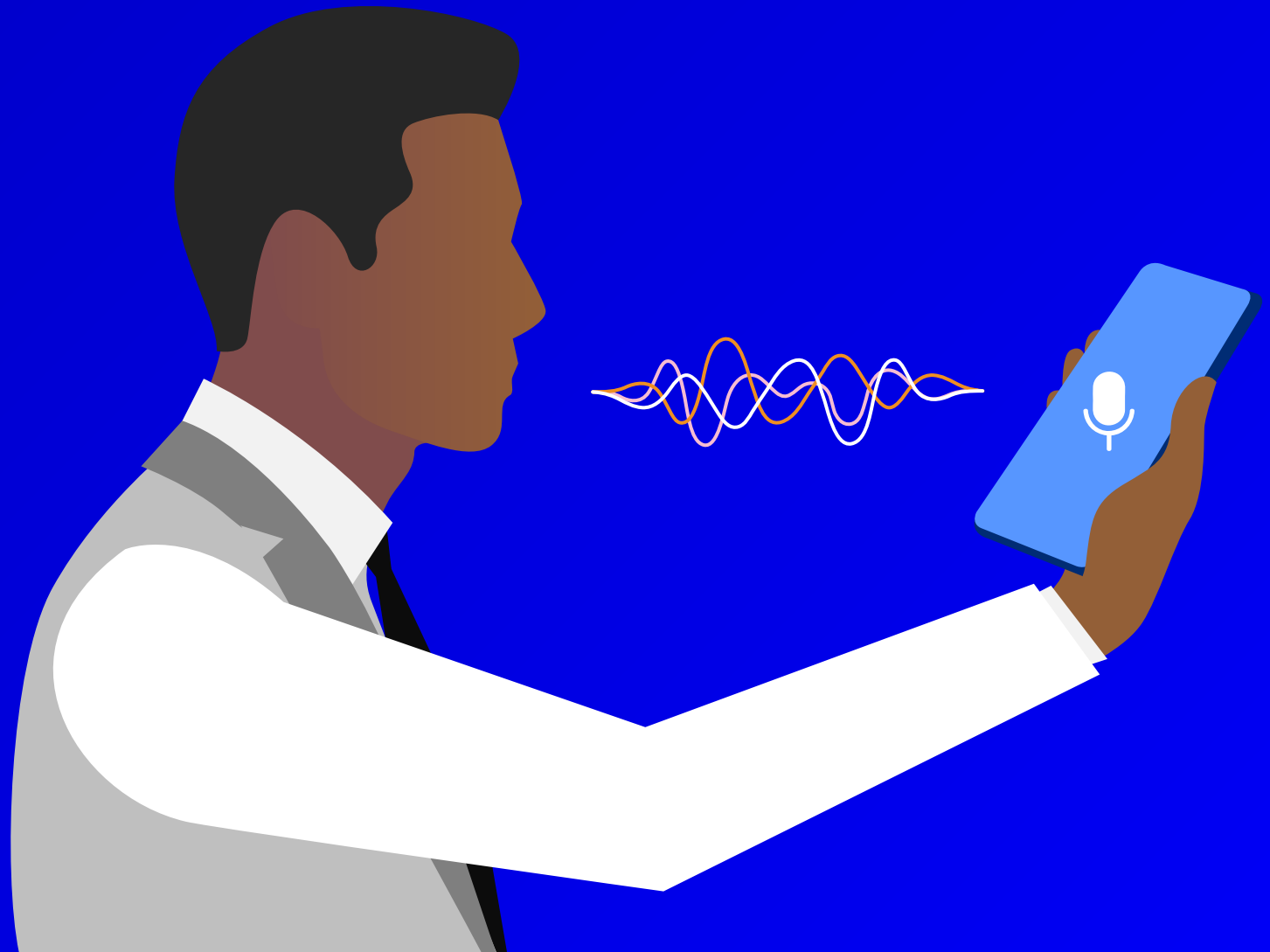
- The paper presents a novel approach for generating adversarial examples in the context of STT systems and their two-step attack offers a perspective on how to generate effective adversarial examples.
- The paper shows their attacks can bypass the state-of-the-art STT models, highlighting the vulnerability of these systems to adversarial attacks.
- The paper discussed some new features of adversarial examples on audio domain.

Strengths and Weaknesses

Weaknesses:

- Limited evaluation. They attacked a specific STT model (DeepSpeech).
- High perturbation needed in order to generate adversarial examples robust to pointwise noise or MP3 compression.
- Does not work over-the-air.
- Limited defense strategies. The papers suggests that improving the robustness of STT models may be an effective defense, but it doesn't explore other defense strategies.

Questions?



Increasing Confidence in Adversarial Robustness Evaluations

Javad Hezareh & Mahdi Saieedi

Security and Privacy in Machine Learning

Sharif University of Technology

Spring 2023

Overview

Increasing Confidence in Adversarial Robustness Evaluations

Roland S. Zimmermann* University of Tübingen
Wieland Brendel University of Tübingen
Florian Tramèr Google
Nicholas Carlini Google

Abstract

Hundreds of defenses have been proposed to make deep neural networks robust against minimal (adversarial) input perturbations. However, only a handful of these defenses held up their claims because correctly evaluating robustness is extremely challenging: Weak attacks often fail to find adversarial examples even if they unknowingly exist, thereby making a vulnerable network look robust.

In this paper, we propose a test to identify weak attacks, and thus weak defense evaluations. Our test slightly modifies a neural network to guarantee the existence of an adversarial example for every sample. Consequentially, any correct attack must succeed in breaking this modified network.

For eleven out of thirteen previously-published defenses, the original evaluation of the defense fails our test, while stronger attacks that break these defenses pass it. We hope that attack *unit tests* — such as ours — will be a major component in future robustness evaluations and increase confidence in an empirical field that is currently riddled with skepticism. Online version & code: zimmerrol.github.io/active-tests/

Figure: Oral at CVPR 2022 Workshop (Art of Robustness), Project website.

Overview

- Can we trust a proposed defense?
- Is defense evaluation valid?

Overview

- Can we trust a proposed defense?
- Is defense evaluation valid?
- Main shame for defense evaluation:
  1. Propose an attack
  2. Evaluate defense with this attack
  3. If no adversarial example found, defense works

Outline

Outline

Theorem Analogy

Lemma 2.4 (Certificate for Empirical Mean)**.** *Let S be an (ϵ, δ)-stable set with respect to a distribution X, for some $\delta \geq \epsilon > 0$. Let T be an ϵ-corrupted version of S. Let μ_T and Σ_T be the empirical mean and covariance of T. If the largest eigenvalue of Σ_T is at most $1 + \lambda$, for some $\lambda \geq 0$, then $\|\mu_T - \mu_X\|_2 \leq O(\delta + \sqrt{\epsilon\lambda})$.*

Proof of Lemma 2.4. Let $S' = S \cap T$ and $T' = T \setminus S'$. By replacing S' with a subset if necessary, we may assume that $|S'| = (1-\epsilon)|S|$ and $|T'| = \epsilon|S|$. Let $\mu_{S'}, \mu_{T'}, \Sigma_{S'}, \Sigma_{T'}$ represent the empirical means and covariance matrices of S' and T'. A simple calculation gives that

$$\Sigma_T = (1-\epsilon)\Sigma_{S'} + \epsilon\Sigma_{T'} + \epsilon(1-\epsilon)(\mu_{S'} - \mu_{T'})(\mu_{S'} - \mu_{T'})^T .$$

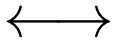
Let v be the unit vector in the direction of $\mu_{S'} - \mu_{T'}$. We have that

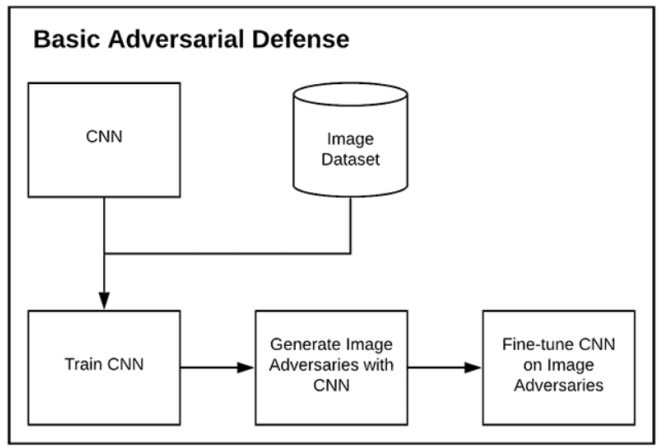
$$\begin{aligned} 1 + \lambda \geq v^T\Sigma_T v &= (1-\epsilon)v^T\Sigma_{S'}v + \epsilon v^T\Sigma_{T'}v + \epsilon(1-\epsilon)v^T(\mu_{S'} - \mu_{T'})(\mu_{S'} - \mu_{T'})^T v \\ &\geq (1-\epsilon)(1 - \delta^2/\epsilon) + \epsilon(1-\epsilon)\|\mu_{S'} - \mu_{T'}\|_2^2 \\ &\geq 1 - O(\delta^2/\epsilon) + (\epsilon/2)\|\mu_{S'} - \mu_{T'}\|_2^2 , \end{aligned}$$

where we used the variational characterization of eigenvalues, the fact that $\Sigma_{T'}$ is positive semidefinite, and the second stability condition for S'. By rearranging, we obtain that $\|\mu_{S'} - \mu_{T'}\|_2 = O(\delta/\epsilon + \sqrt{\lambda/\epsilon})$. Therefore, we can write

$$\begin{aligned} \|\mu_T - \mu_X\|_2 = \|(1-\epsilon)\mu_{S'} + \epsilon\mu_{T'} - \mu_X\|_2 &= \|\mu_{S'} - \mu_X + \epsilon(\mu_{T'} - \mu_{S'})\|_2 \\ &\leq \|\mu_{S'} - \mu_X\|_2 + \epsilon\|\mu_{S'} - \mu_{T'}\|_2 = O(\delta) + \epsilon \cdot O(\delta/\epsilon + \sqrt{\lambda/\epsilon}) \\ &= O(\delta + \sqrt{\lambda\epsilon}) , \end{aligned}$$

where we used the first stability condition for S' and our bound on $\|\mu_{S'} - \mu_{T'}\|_2$. □





Theorem

Proof.

we have proved that $P \neq NP$. $\cdots$ □

- How do you refute the proof's claim?

Theorem

Proof.

we have proved that $P \neq NP$. $\cdots$ □

- How do you refute the proof's claim?
  - Find an algorithm to solve 3-SAT in polynomial time.
  - Studying proofs line-by-line, till find some major flaw

Defense evaluation

Defense X.

We have demonstrated that defense X improves model robustness. One can validate this claim by following below evaluation procedure.
...

- How do you refute the authors' claim?

Defense evaluation

Defense X.

We have demonstrated that defense X improves model robustness. One can validate this claim by following below evaluation procedure.
...

- How do you refute the authors' claim?
  - Find an adversarial attack to decrease model performance
  - Probe defense evaluation, till find some major flaw

Method

- Test attack strength

Method

- Test attack strength
- Design a new task that is solvable by any sufficiently strong attack
  - Injects adversarial examples into a defense
  - Check if the attack is able to find them

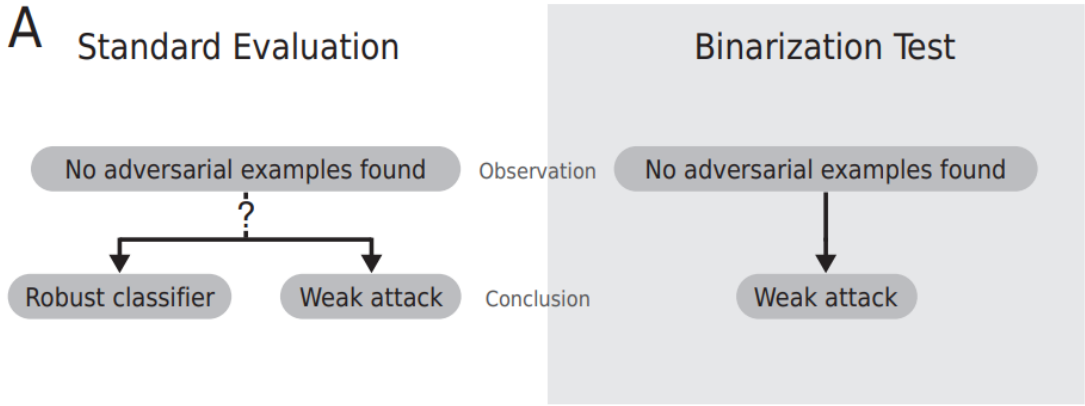


Figure: Proposed method to evaluate the attack used in defense evaluation.

Method

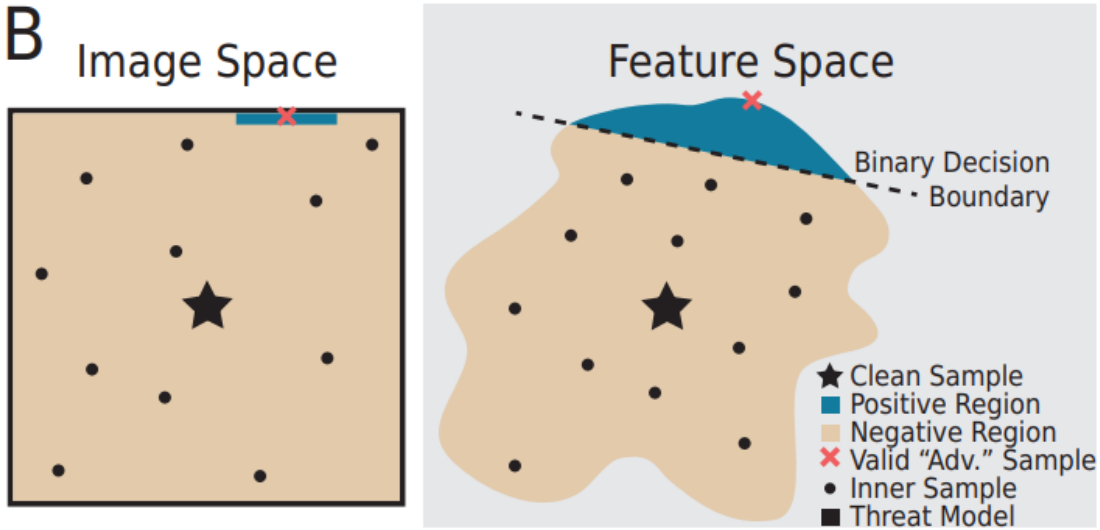


Figure: Inject adversarial examples to check where the attack is powerful enough.

- Rejected attack doesn't necessarily mean the defense is not effective.

Outline

Adversarial Examples

- Imperceptible perturbations that change the decision of a deep neural network in arbitrary directions

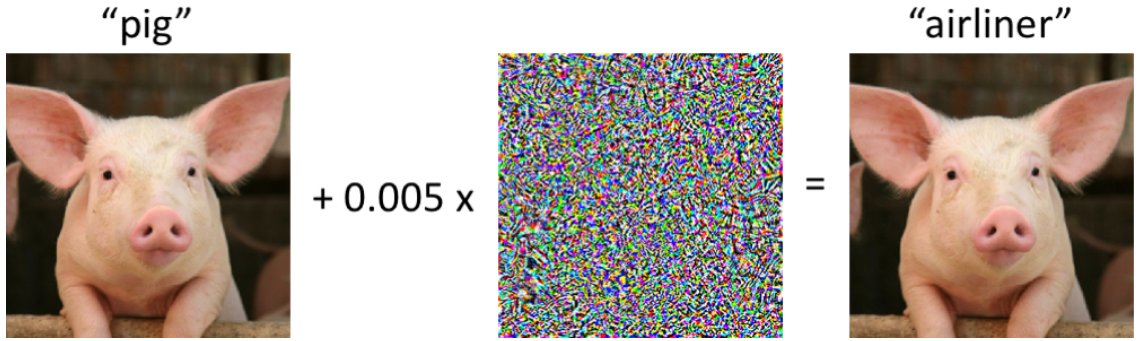


Figure: Adversarial examples.

Defenses

- Add input pre-processing steps
- Introduce architectural changes
- Methods for detecting adversarial examples

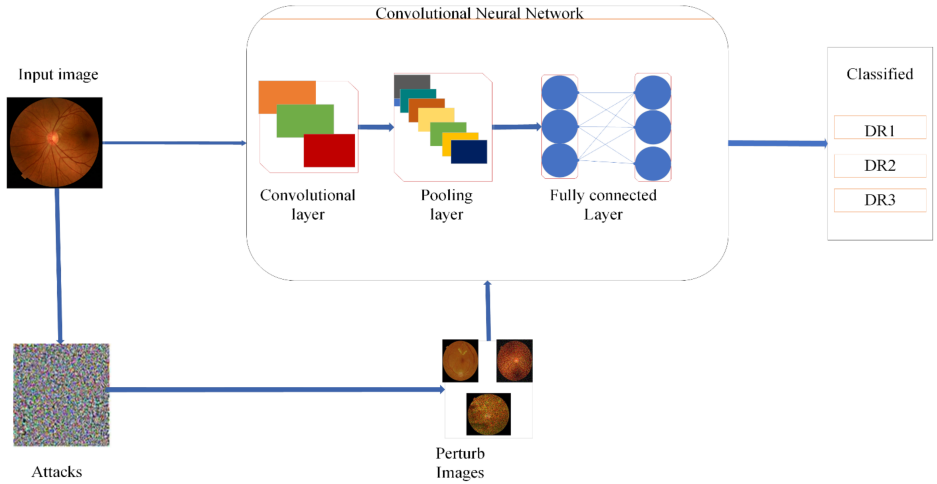


Figure: Adversarial training.

Outline

Defense Evaluation

- Defense evaluation scheme:
  - Perform an attack on defense
  - No adversarial example found within distance $d(\boldsymbol{x}_c, \boldsymbol{x}_{adv}) \leq \epsilon$

Defense Evaluation

- Defense evaluation scheme:
  - Perform an attack on defense
  - No adversarial example found within distance $d(\boldsymbol{x}_c, \boldsymbol{x}_{adv}) \leq \epsilon$
- Attack strength depends on:
  - Attack itself
  - The defense it is meant to evaluate

Outline

Classification Model

- f: our classifier
- f^*: model feature extractor
- $f = f^* +$ linear classification head

New Task Algorithm

- New binary classification task
- Ensure having adversarial examples

Algorithm Build New Task

Require: feature extractor f^* of original classifier, test sample $\boldsymbol{x}_c$, distanse ϵ, number of inner/boundary samples N_i and N_b.

function CreateBinaryClassifier(f^*, $\boldsymbol{x}_c$, ϵ, N_i, N_b)
  $\mathcal{X}_i := \{\boldsymbol{x}_c\} \cup \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) \leq \alpha\epsilon \text{ and } \hat{\boldsymbol{x}} \neq \boldsymbol{x}_c\}_{1:N_i}$
  $\mathcal{X}_b := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \epsilon\}_{1:N_b}$
  $F_i := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_i\}$
  $F_b := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_b\}$
  $\mathcal{D} = \{(\hat{\boldsymbol{x}}, 0) \mid \hat{\boldsymbol{x}} \in F_i\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_b\}$
  $g = \text{TrainLinear}(D)$
  $h = g \circ f^*$
  return h

Binarization Test

Algorithm Binarization Test for Classifiers with Linear Classification Readouts

Require: feature extractor f^* of original classifier, test samples $\mathcal{X}_{test}$, distanse ϵ, number of inner/boundary samples N_i and N_b.

function BinarizationTest(f^*, $\mathcal{X}_{test}$, ϵ, N_i, N_b)
    attack_successful = []
    random_attack_successful = []
    for all $\boldsymbol{x}_c \in \mathcal{X}_{test}$ **do**
        h = CreatBinaryClassifier(f^*, $\boldsymbol{x}_c$, ϵ, N_i, N_b)
        attack_successful.append(Attack($h, \boldsymbol{x}_c$))
        random_attack_successful.append(RandomAttack($h, \boldsymbol{x}_c$))
    ASR = Mean(attack_successful)
    RASR = Mean(random_attack_successful)
    return ASR, RASR

Evaluate New Task

- The efficacy of the used evaluation method:
  - Use original attack to attack h
  - ASR returns *test score*

Evaluate New Task

- The efficacy of the used evaluation method:
  - Use original attack to attack h
  - ASR returns *test score*
- Test difficulty:
  - Use randomized attack
  - #samples = #queris
  - RASR returns this metric

Outline

Models Leveraging Detectors

- Beside classifier f, we have a detector d
- d detects adversarial examples
- A successful attack must fool both the classifier and the defector

Models Leveraging Detectors

- Beside classifier f, we have a detector d
- d detects adversarial examples
- A successful attack must fool both the classifier and the defector
- Two test for this type:
  - Regular Test
  - Inverted Test
- A reliable evaluation must pass both test

Regular Test

Algorithm Build New Task for Classifiers with Detectors

Require: feature extractor f^* of original classifier, adversarial detector d, test sample $\boldsymbol{x}_c$, distanse ϵ, number of inner/boundary/reference samples $N_i/N_b/N_r$.

function CreateBinaryClassifier(f^*, d, $\boldsymbol{x}_c$, ϵ, N_i, N_b, N_r)
  $\mathcal{X}_i := \{\boldsymbol{x}_c\} \cup \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) \leq \alpha\epsilon \text{ and } \hat{\boldsymbol{x}} \neq \boldsymbol{x}_c\}_{1:N_i}$
  $\mathcal{X}_b := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \epsilon, d(\hat{\boldsymbol{x}}) = 1\}_{1:N_b}$
  $\mathcal{X}_r := \{\hat{\boldsymbol{x}} \mid d(\boldsymbol{x}_c, \hat{\boldsymbol{x}}) = \eta\epsilon, d(\hat{\boldsymbol{x}}) = 1\}_{1:N_r}$
  $F_i := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_i\}$
  $F_b := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_b\}$
  $F_r := \{f^*(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}_r\}$
  $\mathcal{D} = \{(\hat{\boldsymbol{x}}, 0) \mid \hat{\boldsymbol{x}} \in F_i\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_b\} \cup \{(\hat{\boldsymbol{x}}, 1) \mid \hat{\boldsymbol{x}} \in F_r\}$
  $b = \text{TrainLinear}(D)$
  return b, $\mathcal{X}_r$

Binarization Test For Detectors

Algorithm Binarization Test for Classifiers with Linear Classification Readouts and a Detector

Require: feature extractor f^* of original classifier, adversarial detector d, test samples $\mathcal{X}_{test}$, distanse ϵ, number of inner/boundary/reference samples $N_i/N_b/N_r$.

function BinarizationTest(f^*, d, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, η)
    attack_successful = []
    random_attack_successful = []
    for all $\boldsymbol{x}_c \in \mathcal{X}_{test}$ **do**
        $b, \mathcal{X}_r$ = CreatBinaryClassifier(f^*, d, $\boldsymbol{x}_c$, ϵ, N_i, N_b, N_r)
        attack_successful.append(Attack($b, d, \boldsymbol{x}_c, \mathcal{X}_r$)
        random_attack_successful.append(RandomAttack($b, d, \boldsymbol{x}_c, \mathcal{X}_r$)
    ASR = Mean(attack_successful)
    RASR = Mean(random_attack_successful)
    return ASR, RASR

Inverted Test

- Why do we need the inverted test?

Algorithm Inverted Test

1: **function** InvertedBinarizationTest(f^*, d, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, ϵ, η)
2: **return** BinarizationTest(f^*, $\neg d$, $\mathcal{X}_{test}$, ϵ, N_i, N_b, N_r, ϵ, η)

Outline

Evaluations

- Defenses without Detectors
- Defenses with Detectors

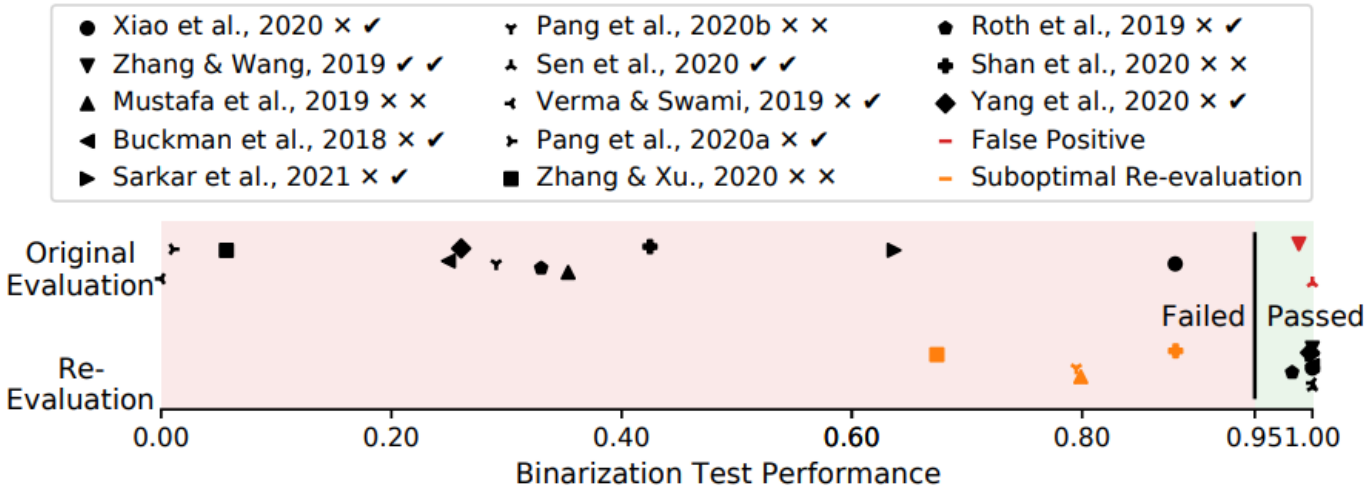


Figure: Binarization Test result for 13 Defenses

Evaluation

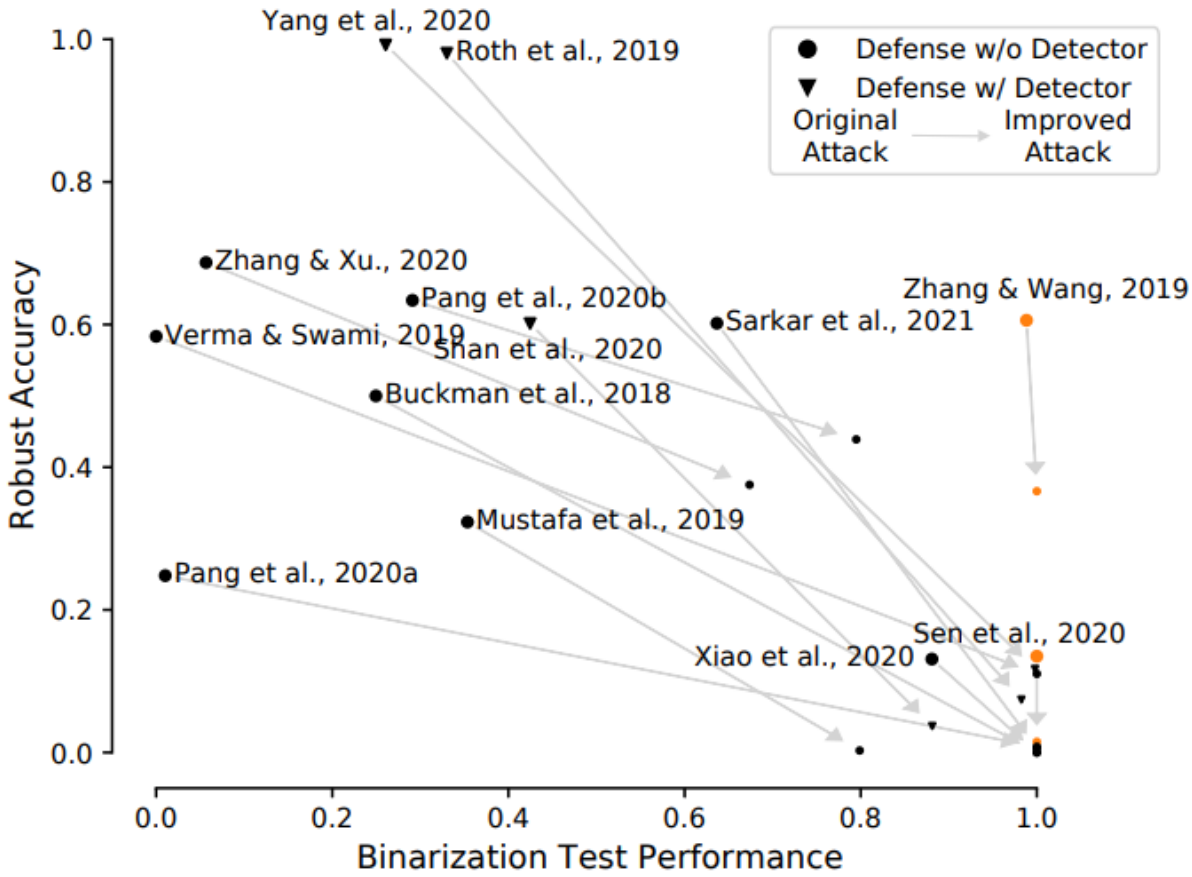


Figure: Robust accuracy as a function of the test performance.

Hardness of Test

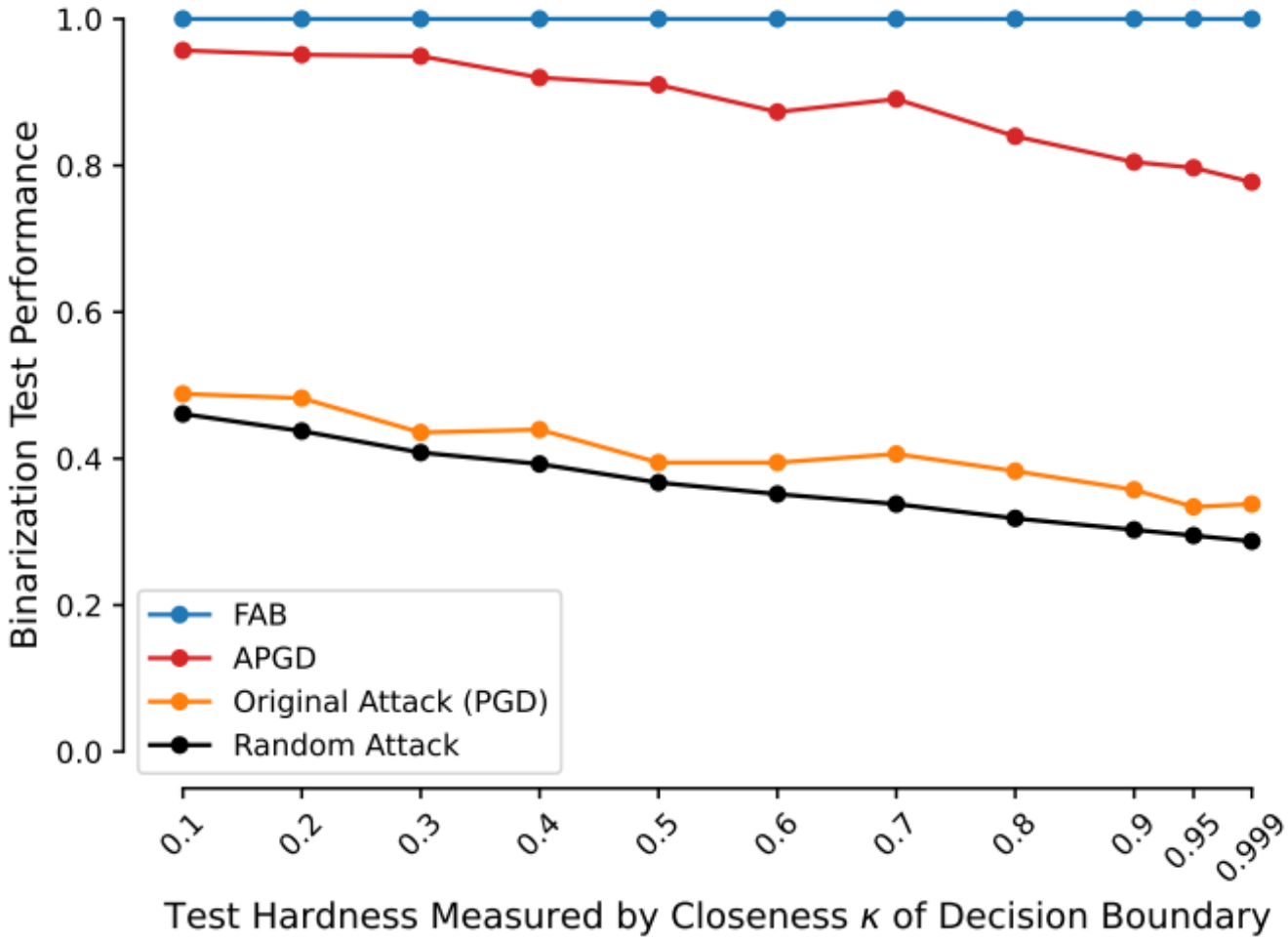


Figure: Hyperparameters influence the test's hardness

Thanks

Any questions?