Adversarial Attacks

Deep Learning

Ali Saffouri [redacted] [redacted]

University of Houston

April, 2024

Introduction

Deep Neural Networks are susceptible to adversarial attacks.

Introduction

Deep Neural Networks are susceptible to adversarial attacks.

Adversarial attacks are inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence.

What is Adversarial Machine Learning?

Adversarial machine learning is a machine learning method that aims to trick machine learning models by providing deceptive input.

What is Adversarial Machine Learning?

Adversarial machine learning is a machine learning method that aims to trick machine learning models by providing deceptive input.

This includes both the generation and detection of adversarial examples, which are inputs specially created to deceive classifiers.

What is Adversarial Machine Learning?

Adversarial machine learning is a machine learning method that aims to trick machine learning models by providing deceptive input.

This includes both the generation and detection of adversarial examples, which are inputs specially created to deceive classifiers.

This method has been extensively explored in areas like spam detection and image classification, where modifications are performed on images to produce incorrect predictions

Literature Review

Explaining and Harnessing Adversarial Examples, by Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

Literature Review

Explaining and Harnessing Adversarial Examples, by Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

1. Adversarial examples can be explained as a property of high-dimensional dot products. They are a result of models being too linear, rather than too nonlinear.

- Previous literature theorized that adversarial examples were due to extreme nonlinearity of deep neural networks, combined with insufficient model averaging and insufficient regularization of the purely supervised learning problem
- However, Goodfellow et al. demonstrate that linear behavior in high dimension spaces is sufficient for designing a fast method of generating adversarial examples to make adversarial training practical

Literature Review

Explaining and Harnessing Adversarial Examples, by Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

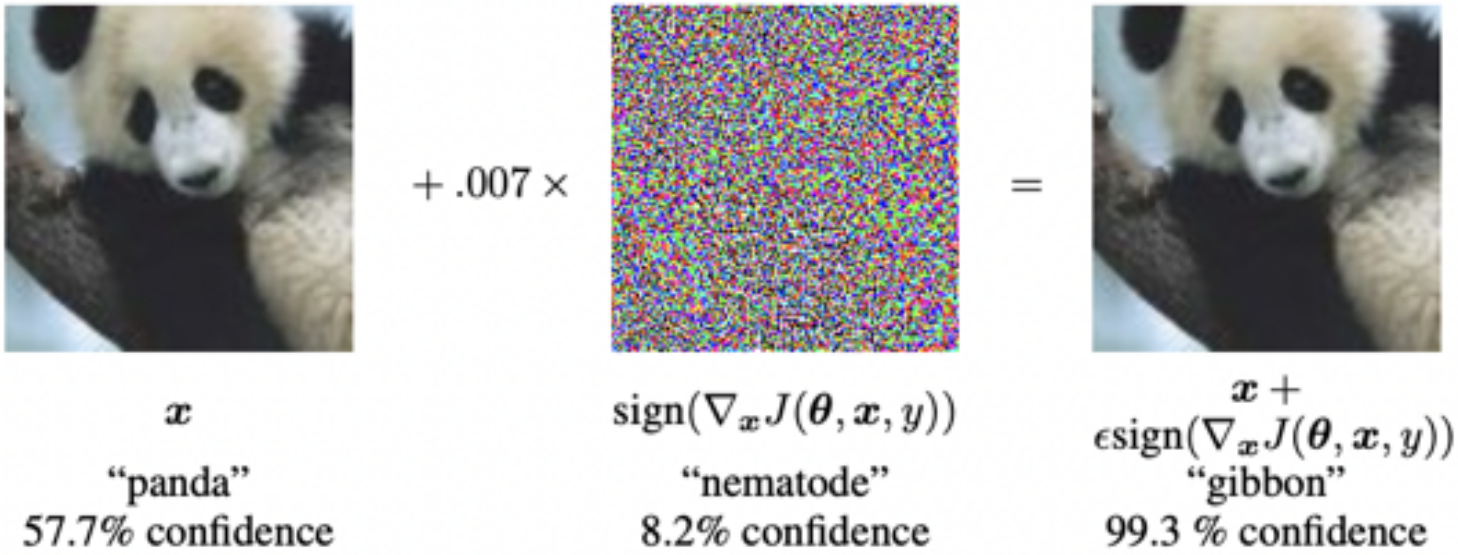
2. Fast Gradient Sign Method

- Let θ be the parameters of a model, x the input to the model, y the targets associated with x (for machine learning tasks that have targets), and $J(\theta, x, y)$ be the cost used to train the neural network. We can linearize the cost function around the current value of θ, obtaining an optimal max-norm constrained perturbation of

$$\eta = \text{sign}\left(\nabla_x J(\theta, x, y)\right)$$

Literature Review

2. Fast Gradient Sign Method



- By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image

Literature Review

3. The generalization of adversarial examples across different models can be explained as a result of adversarial perturbations being highly aligned with the weight vectors of a model, and different models learning similar functions when trained to perform the same task.

4. The direction of perturbation, rather than the specific point in space, matters most.

5. Models that are easy to optimize are easy to perturb

Literature Review

6. Adversarial Training

- Can result in regularization, even further regularization than drop out
- Linear models lack the capacity to resist adversarial perturbation; only structures with a hidden layer (where the universal approximator theorem applies) should be trained to resist adversarial perturbation.
- By exposing the model to these adversarial examples during training, it learns to be more robust to such perturbations and improves its generalization performance.

Literature Review

7. Broader Implications and Future Directions

- Broader implications besides image classification include natural language processing and reinforcement learning
- Understanding the vulnerability of machine learning models to adversarial examples is crucial for deploying reliable and trustworthy AI systems in safety-critical applications
- Challenges remain in scaling it to complex models, as it can be computationally expensive
- Future research directions include developing more efficient defense mechanisms, understanding the theoretical properties of adversarial examples, and exploring new paradigms for designing robust and interpretable machine learning models

Attacking Pretrained Models

- We perform adversarial attacks on a subset of classes from ImageNet, as opposed to the literature using MNIST, to avoid retraining any models, as the models we obtained were all pretrained on ImageNet. We used the validation set provided by imagenette.
  - 10 Classes: tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute
  - Imagenette Validation Set Size: 3925
  - We reclassify the classes according to their original ImageNet classification
- We followed a tutorial on PyTorch applying FGSM on MNIST
- We only attack models on images they classify correctly.

Attacking Pretrained Models

Model	Weights	Acc@1	Acc@5	Params	GFLOPS
GoogLeNet	GoogLeNet_Weights.IMAGENET1K_V1	69.778	89.53	6.6M	1.5
VGG11	VGG11_Weights.IMAGENET1K_V1	69.02	88.628	132.9M	7.61
Wide ResNet	Wide_ResNet50_2_Weights.IMAGENET1K_V2	81.602	95.758	68.9M	11.4
ConvNext	ConvNeXt_Tiny_Weights.IMAGENET1K_V1	82.52	96.146	28.6M	4.46

```python
def fgsm_attack(image, epsilon, data_grad):
    # Collect the element-wise sign of the data gradient
    sign_data_grad = data_grad.sign()
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

Retraining VGG11

- To improve the robustness of VGG11 to adversarial attacks, we partially retrain VGG11.
  1. We generated perturbed images on all imagenette's training set (size: 9469) by attacking pretrained VGG11 using FGSM on epsilon 0.07
  2. We froze the VGG11 feature extractor and retrained the classifier on the clean and perturbed images. The classifier architecture was not modified.
  3. Early stopping was implemented if the validation loss did not decrease for 3 consecutive epochs.
- We test the robustness by following our previous procedure to attack models

Sample Adversarial Attack Pipeline

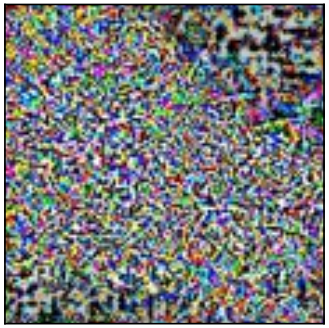
Original Image

Image Transformed

Transformed Image Denormalized

Denormalized Image Perturbed

Noise



Purturbed Image Normalized

Model: GoogLeNet, Epsilon 0.007: chain_saw, 0.93 -> muzzle, 0.82

Sample Adversarial Attack

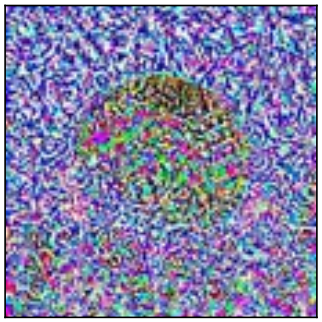
Original Image

Image Transformed

Transformed Image Denormalized

Denormalized Image Perturbed

Noise



Purturbed Image Normalized

Model: VGG, Epsilon 0.07: golf_ball, 1.00 -> baseball, 0.49

Sample Adversarial Attack

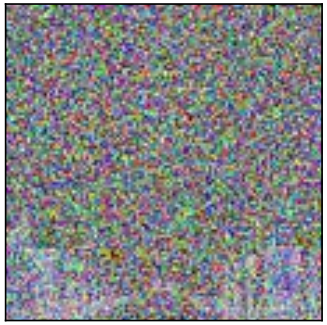
Original Image

Image Transformed

Transformed Image Denormalized

Denormalized Image Perturbed

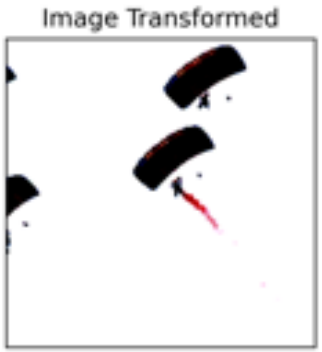
Noise

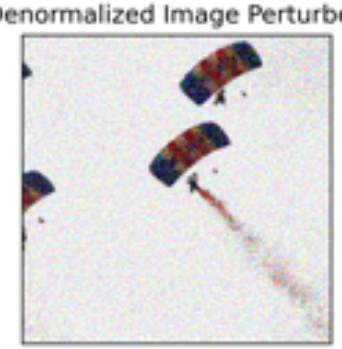


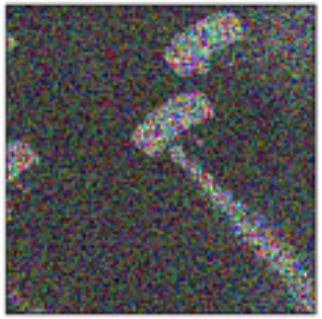
Purturbed Image Normalized

Model: ConvNeXt, Epsilon 0.15: church, 0.86 -> vault, 0.35

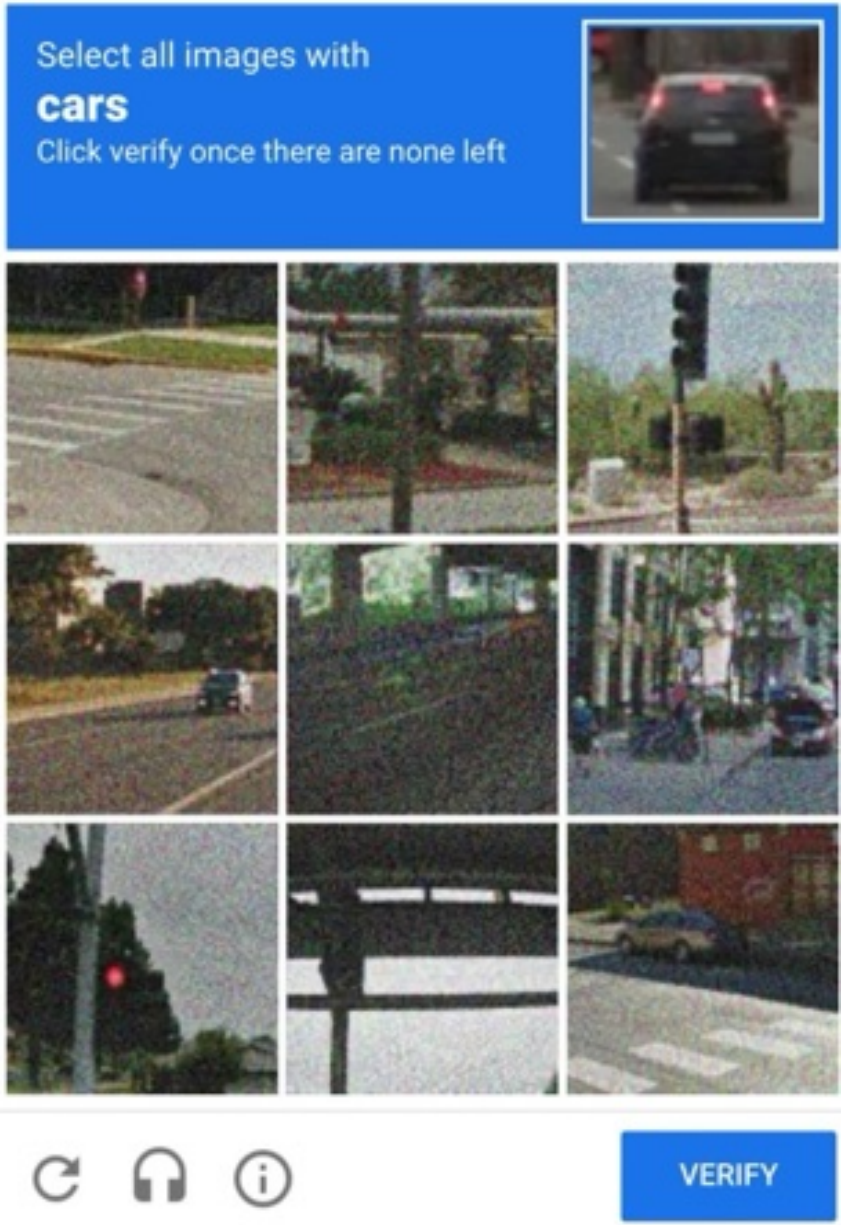
Sample Failed Adversarial Attack



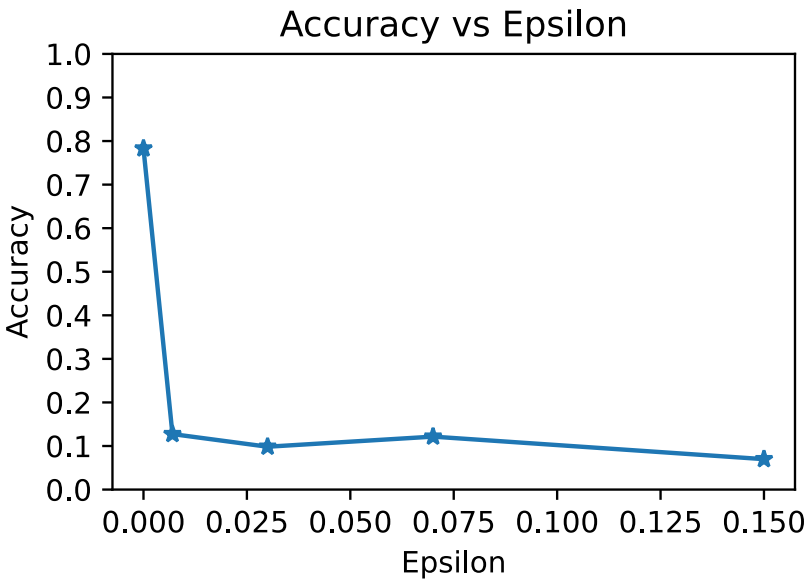


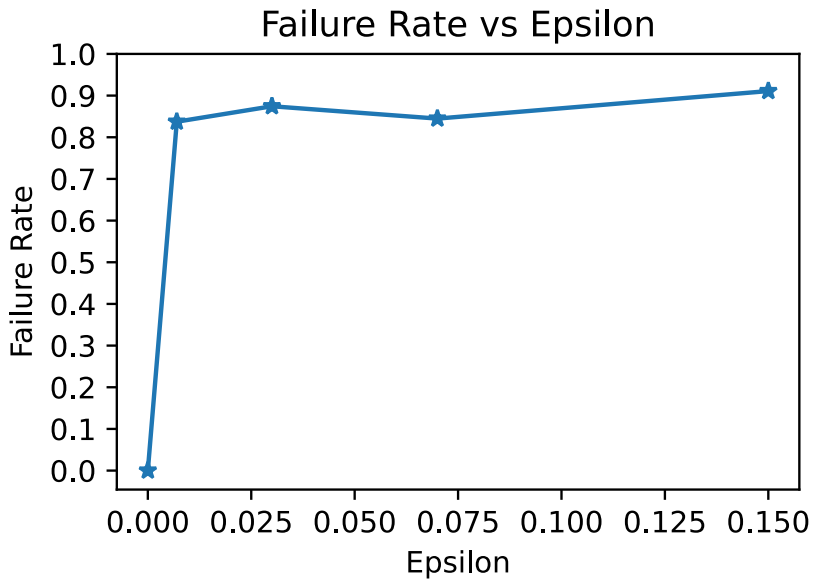


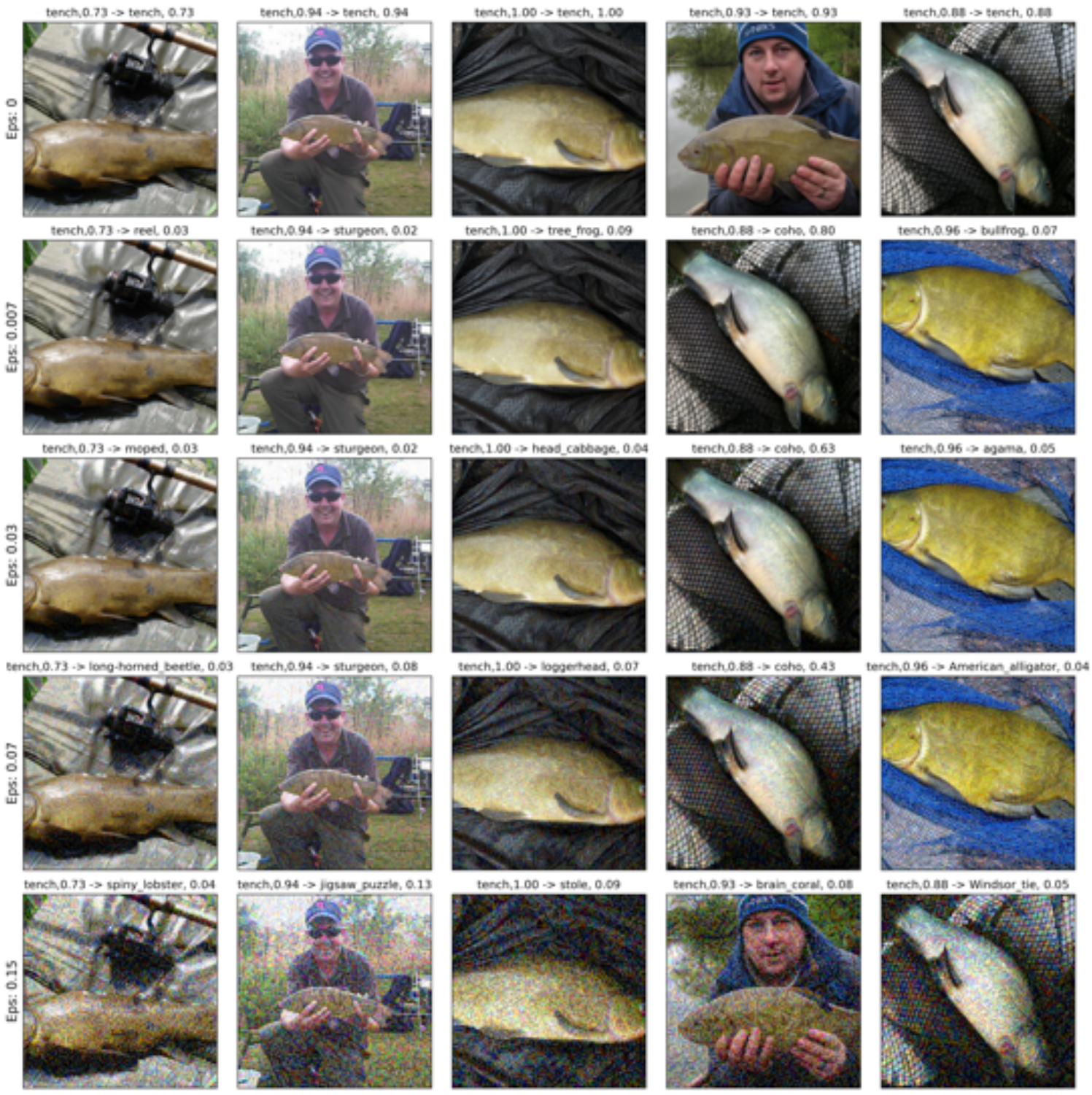
Model: ConvNeXt, Epsilon 0.15: parachute, 0.95 -> parachute, 0.89


Select all images with
cars
Click verify once there are none left
VERIFY

GoogleNet

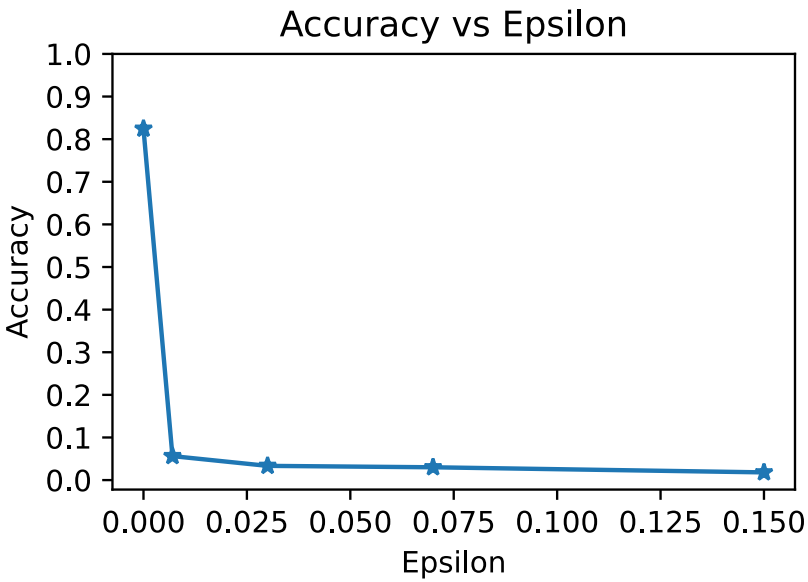

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150

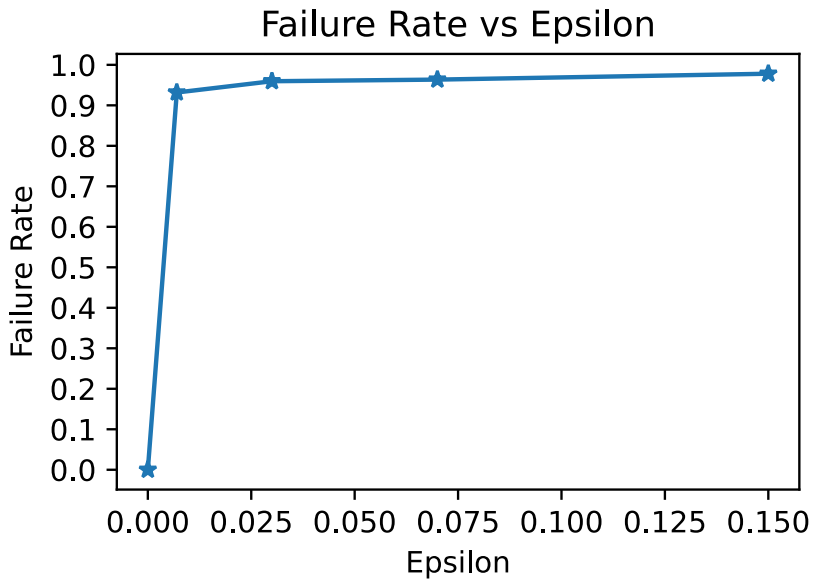

Failure Rate vs Epsilon
Failure Rate
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150


Eps: 0
tench,0.73 -> tench, 0.73
tench,0.94 -> tench, 0.94
tench,1.00 -> tench, 1.00
tench,0.93 -> tench, 0.93
tench,0.88 -> tench, 0.88
Eps: 0.007
tench,0.73 -> reel, 0.03
tench,0.94 -> sturgeon, 0.02
tench,1.00 -> tree_frog, 0.09
tench,0.88 -> coho, 0.80
tench,0.96 -> bullfrog, 0.07
Eps: 0.03
tench,0.73 -> moped, 0.03
tench,0.94 -> sturgeon, 0.02
tench,1.00 -> head_cabbage, 0.04
tench,0.88 -> coho, 0.63
tench,0.96 -> agama, 0.05
Eps: 0.07
tench,0.73 -> long-horned_beetle, 0.03
tench,0.94 -> sturgeon, 0.08
tench,1.00 -> loggerhead, 0.07
tench,0.88 -> coho, 0.43
tench,0.96 -> American_alligator, 0.04
Eps: 0.15
tench,0.73 -> spiny_lobster, 0.04
tench,0.94 -> jigsaw_puzzle, 0.13
tench,1.00 -> stole, 0.09
tench,0.93 -> brain_coral, 0.08
tench,0.88 -> Windsor_tie, 0.05

GoogleNet

VGG11

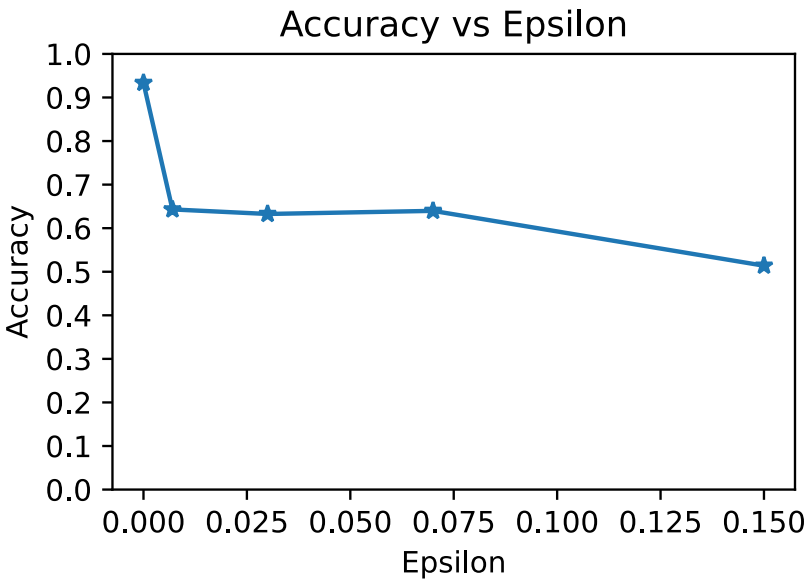

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150

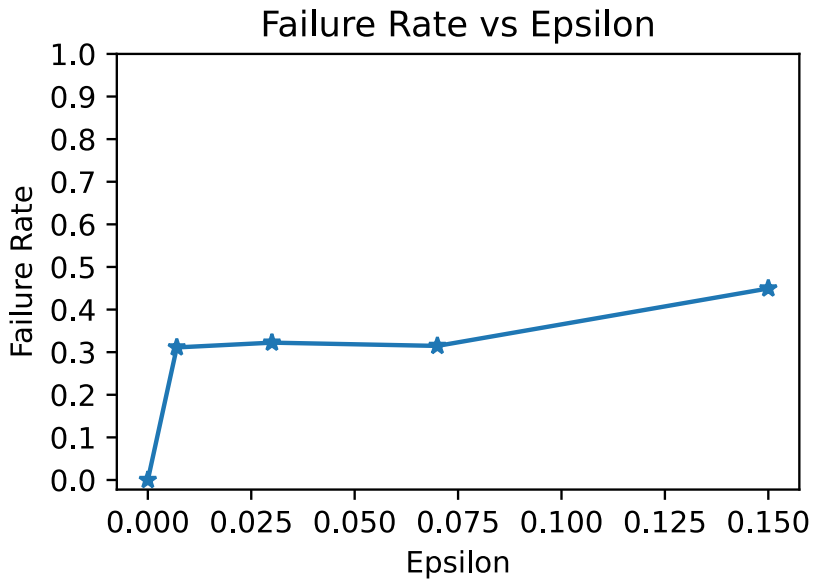

Failure Rate vs Epsilon
Failure Rate
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150


Eps: 0
tench,0.98 -> tench, 0.98
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
tench,0.93 -> tench, 0.93
Eps: 0.007
tench,0.98 -> rifle, 0.25
tench,1.00 -> barracouta, 0.47
tench,1.00 -> stole, 0.17
tench,1.00 -> barracouta, 0.92
tench,0.93 -> coho, 1.00
Eps: 0.03
tench,0.98 -> rifle, 0.20
tench,1.00 -> gar, 0.35
tench,1.00 -> stole, 0.29
tench,1.00 -> barracouta, 0.60
tench,0.93 -> coho, 0.92
Eps: 0.07
tench,0.98 -> rifle, 0.18
tench,1.00 -> water_buffalo, 0.09
tench,1.00 -> stole, 0.25
tench,1.00 -> vestment, 0.22
tench,0.93 -> coho, 0.48
Eps: 0.15
tench,0.98 -> screw, 0.05
tench,1.00 -> jigsaw_puzzle, 0.23
tench,1.00 -> prayer_rug, 0.27
tench,1.00 -> coil, 0.06
tench,0.93 -> chainlink_fence, 0.47

VGG11

Wide ResNet

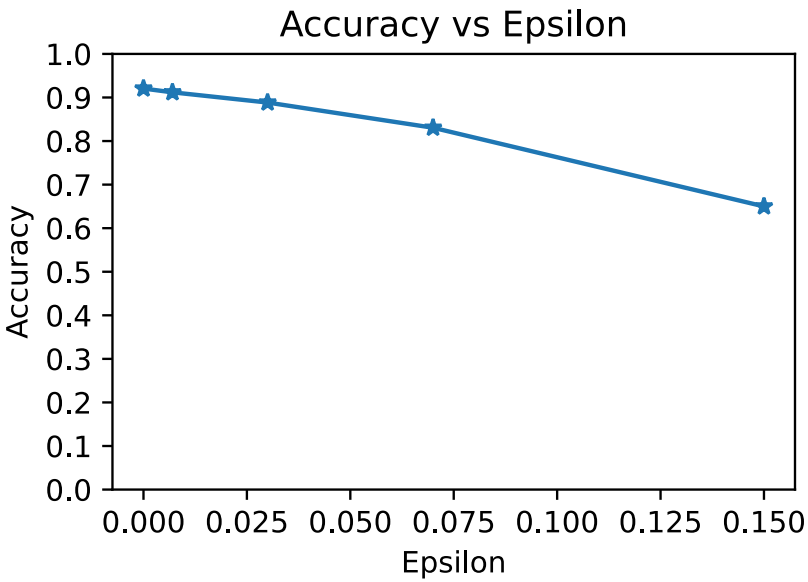

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150

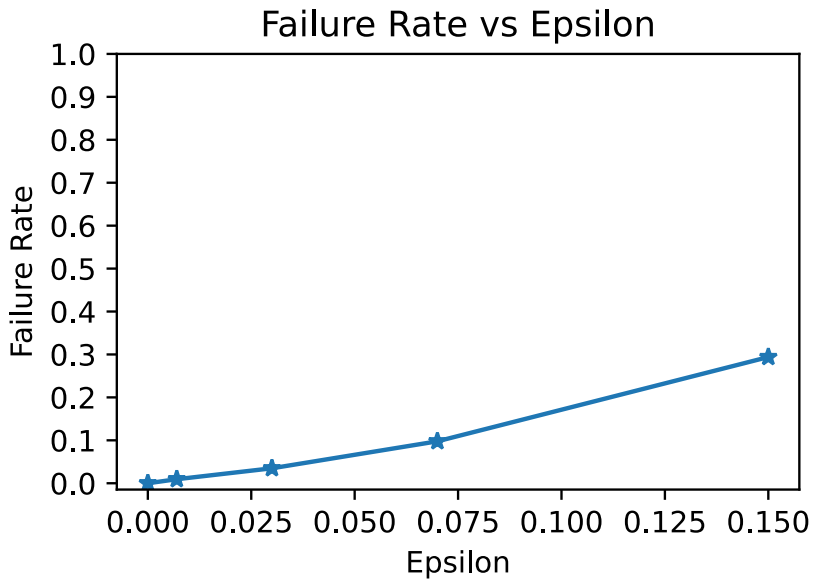

Failure Rate vs Epsilon
Failure Rate
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150

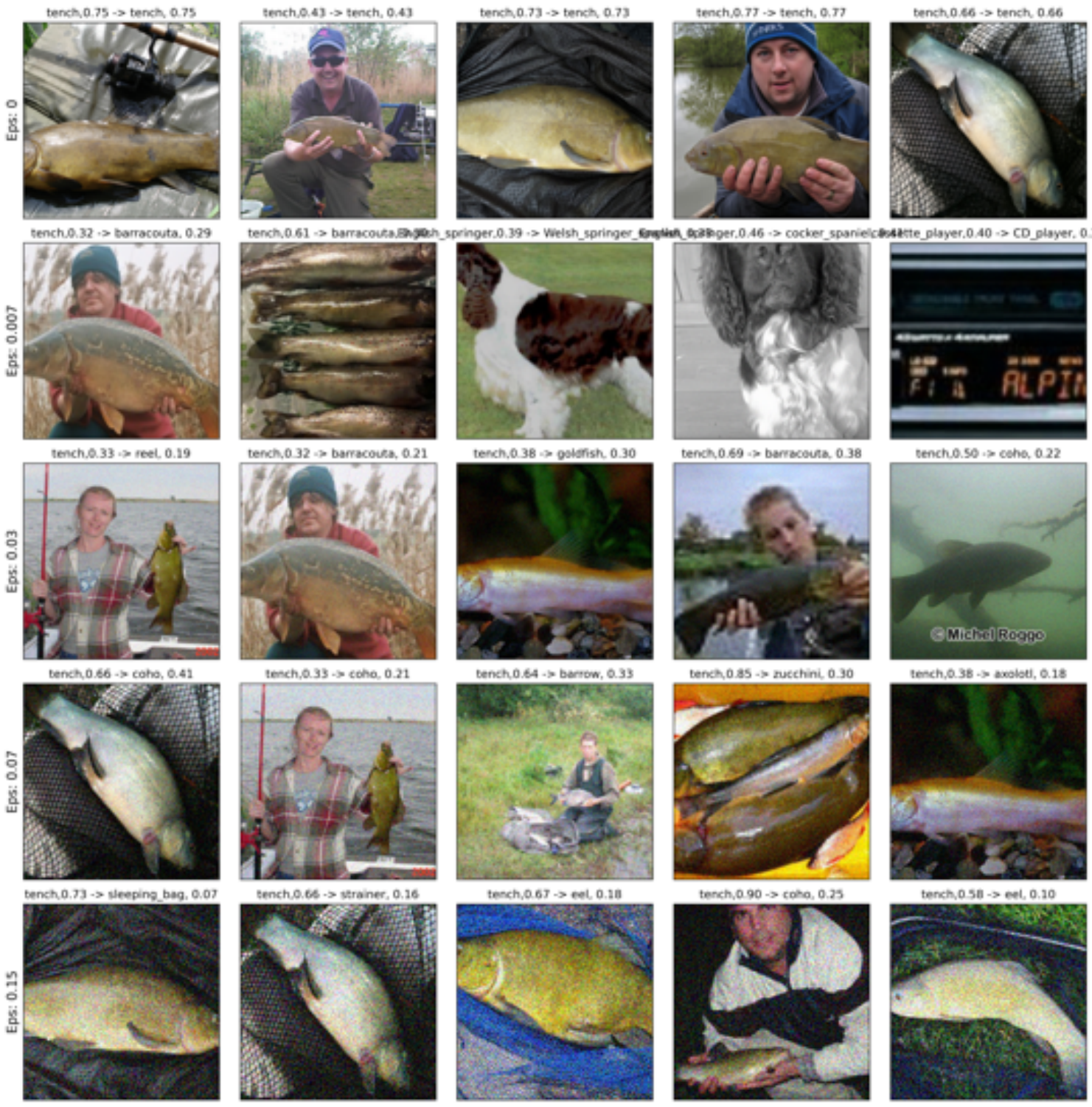

Eps: 0
tench,0.62 -> tench, 0.62
tench,0.51 -> tench, 0.51
tench,0.39 -> tench, 0.39
tench,0.53 -> tench, 0.53
tench,0.41 -> tench, 0.41
Eps: 0.007
tench,0.41 -> coho, 0.02
tench,0.34 -> barracouta, 0.12
tench,0.56 -> barracouta, 0.29
tench,0.51 -> barracouta, 0.33
tench,0.76 -> goose, 0.03
Eps: 0.03
tench,0.41 -> coho, 0.04
tench,0.34 -> barracouta, 0.15
tench,0.56 -> barracouta, 0.28
tench,0.51 -> barracouta, 0.50
tench,0.76 -> coho, 0.07
Eps: 0.07
tench,0.41 -> coho, 0.45
tench,0.48 -> barracouta, 0.30
tench,0.52 -> sock, 0.17
tench,0.34 -> maillot, 0.06
tench,0.56 -> wombat, 0.08
Eps: 0.15
tench,0.62 -> sleeping_bag, 0.05
tench,0.39 -> sleeping_bag, 0.07
tench,0.41 -> electric_fan, 0.04
tench,0.73 -> bow_tie, 0.11
tench,0.48 -> barracouta, 0.33

Wide ResNet

ConvNeXt

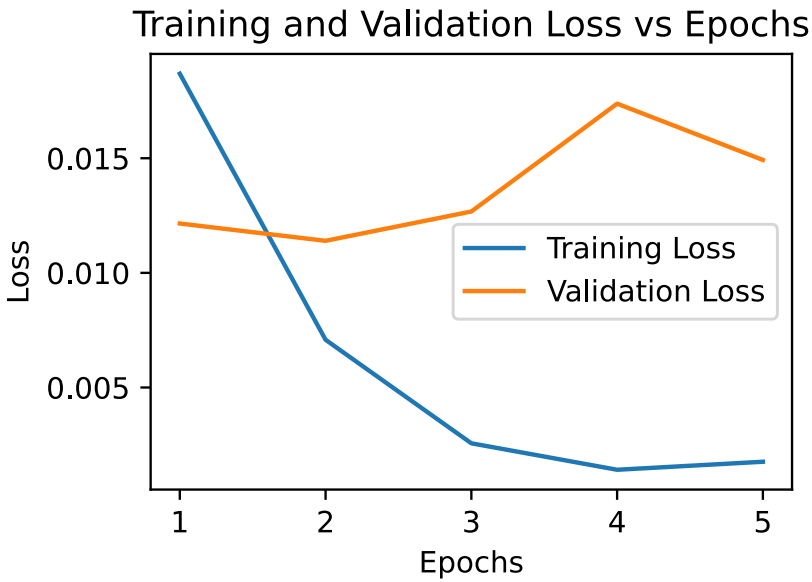

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150

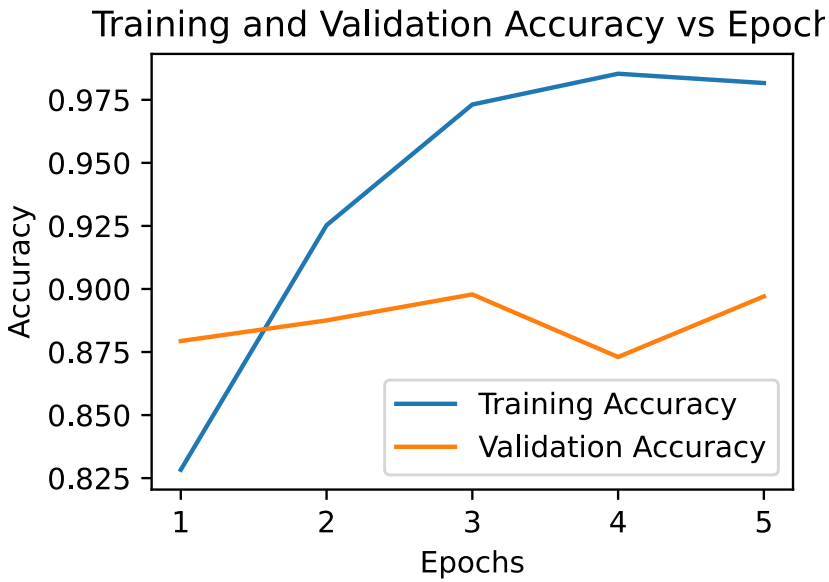

Failure Rate vs Epsilon
Failure Rate
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150


Eps: 0
tench,0.75 -> tench, 0.75
tench,0.43 -> tench, 0.43
tench,0.73 -> tench, 0.73
tench,0.77 -> tench, 0.77
tench,0.66 -> tench, 0.66
Eps: 0.007
tench,0.32 -> barracouta, 0.29
tench,0.61 -> barracouta, 0.30
Welsh_springer,0.39 -> Welsh_springer,0.46 -> cocker_spaniel,0.40 -> CD_player, 0.39
Eps: 0.03
tench,0.33 -> reel, 0.19
tench,0.32 -> barracouta, 0.21
tench,0.38 -> goldfish, 0.30
tench,0.69 -> barracouta, 0.38
tench,0.50 -> coho, 0.22
© Michel Roggo
Eps: 0.07
tench,0.66 -> coho, 0.41
tench,0.33 -> coho, 0.21
tench,0.64 -> barrow, 0.33
tench,0.85 -> zucchini, 0.30
tench,0.38 -> axolotl, 0.18
Eps: 0.15
tench,0.73 -> sleeping_bag, 0.07
tench,0.66 -> strainer, 0.16
tench,0.67 -> eel, 0.18
tench,0.90 -> coho, 0.25
tench,0.58 -> eel, 0.10

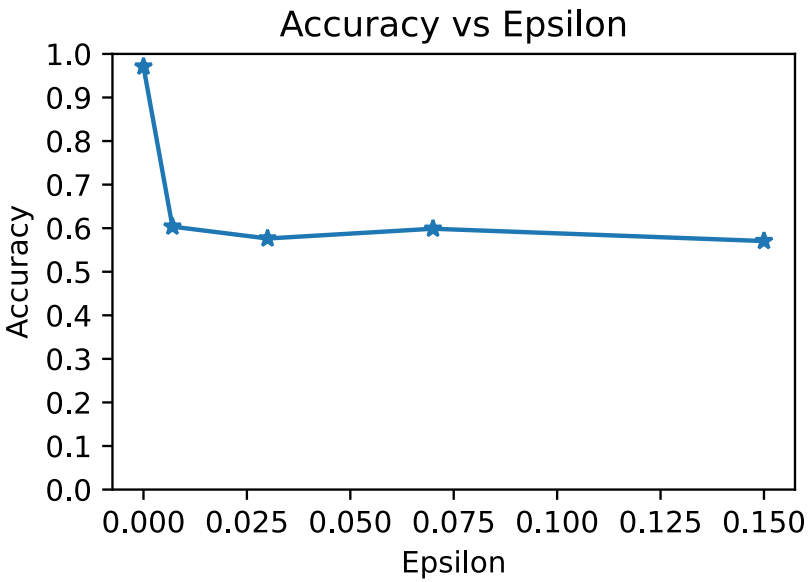
ConvNeXt

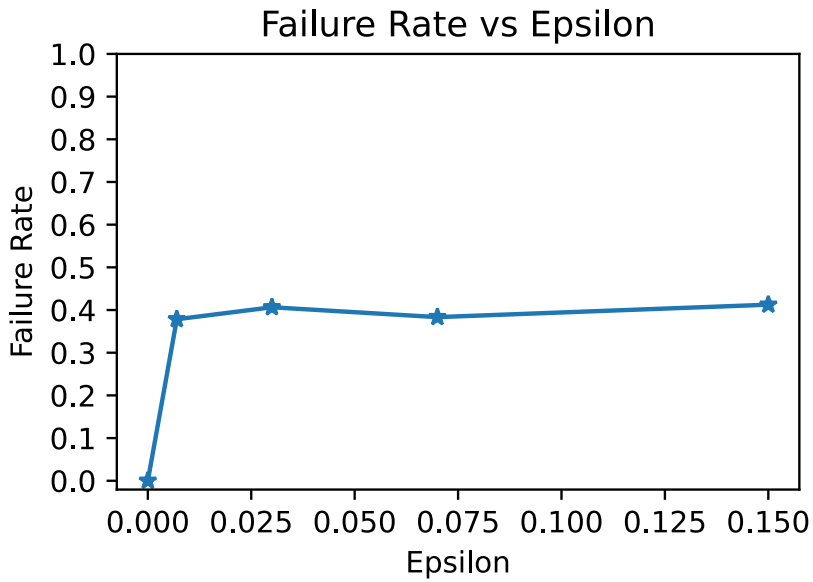
Retraining VGG11

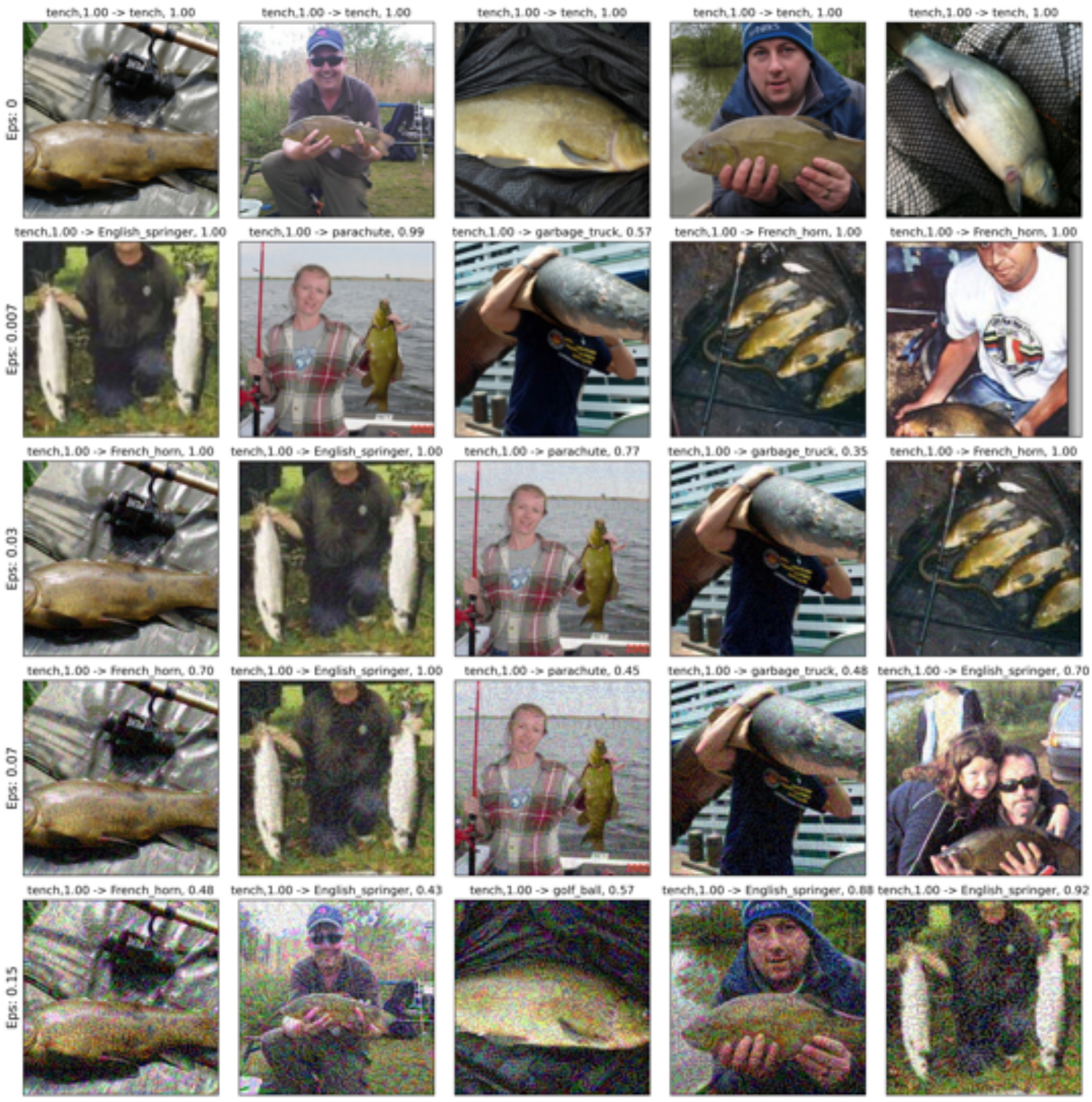




Attacking Retrained VGG11






Eps: 0
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
tench,1.00 -> tench, 1.00
Eps: 0.007
tench,1.00 -> English_springer, 1.00
tench,1.00 -> parachute, 0.99
tench,1.00 -> garbage_truck, 0.57
tench,1.00 -> French_horn, 1.00
tench,1.00 -> French_horn, 1.00
Eps: 0.03
tench,1.00 -> French_horn, 1.00
tench,1.00 -> English_springer, 1.00
tench,1.00 -> parachute, 0.77
tench,1.00 -> garbage_truck, 0.35
tench,1.00 -> French_horn, 1.00
Eps: 0.07
tench,1.00 -> French_horn, 0.70
tench,1.00 -> English_springer, 1.00
tench,1.00 -> parachute, 0.45
tench,1.00 -> garbage_truck, 0.48
tench,1.00 -> English_springer, 0.70
Eps: 0.15
tench,1.00 -> French_horn, 0.48
tench,1.00 -> English_springer, 0.43
tench,1.00 -> golf_ball, 0.57
tench,1.00 -> English_springer, 0.88
tench,1.00 -> English_springer, 0.92

VGG11

Conclusions

- GoogleNet and VGG are not robust to the FGSM adversarial attack. Even with very minimal perturbation that are not noticeable to the human eye.
- Models more robust to FGSM, are still fooled with sufficient perturbations.
- Retraining models with adversarial images improves its robustness.

Robustness of Retrained VGG11 on New Classes

Does the Retrained VGG11 model retain accuracy on classes not included in the test set using a different subset of ImageNet obtained from Kaggle validation set that includes 100 ImageNet classes and has a total of 3923 images.

Epsilon: 0.007, Test Accuracy = 29 / 3923 = 0.007, Failure Rate = 122 / 151 = 0.808

Transferability of Adversarial Examples

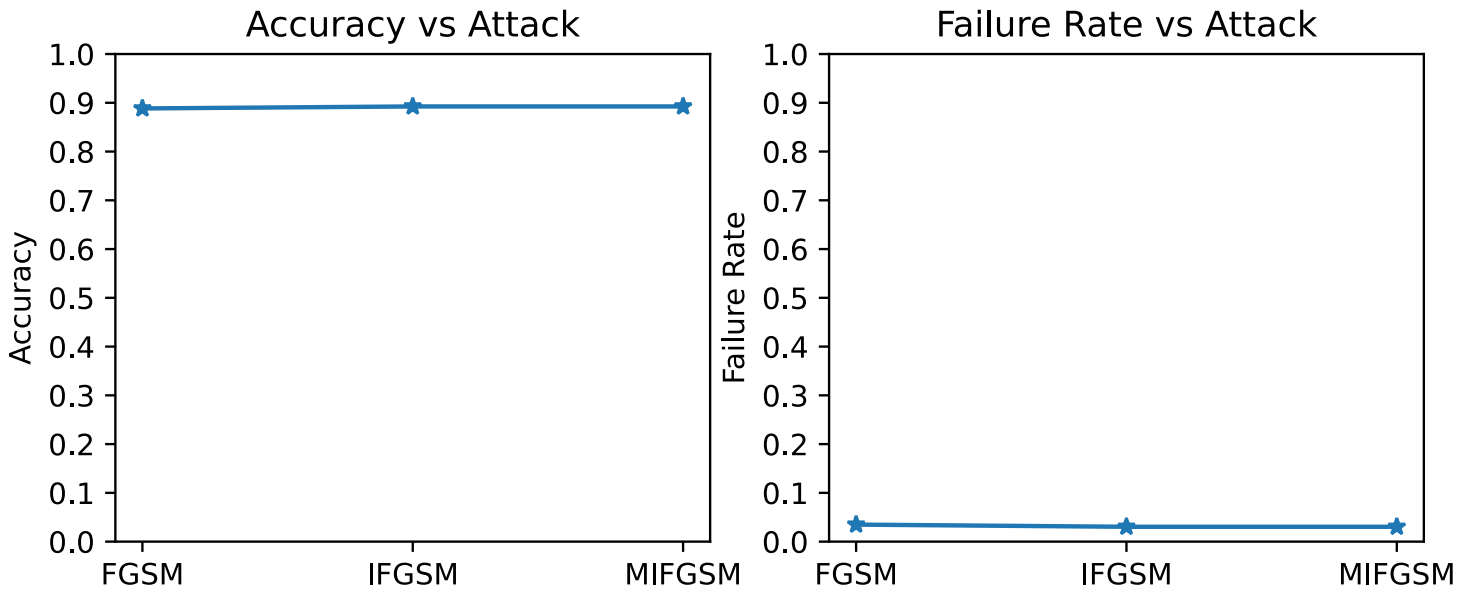
Can adversarial examples generated for VGG11 also deceive the other models?

Yes:

Model Name: GoogLeNet, Test Accuracy = 3213 / 9469 = 0.339
Model Name: VGG, Test Accuracy = 229 / 9469 = 0.024
Model Name: ResNet, Test Accuracy = 4997 / 9469 = 0.528
Model Name: ConvNeXt, Test Accuracy = 4158 / 9469 = 0.439

Different Attacks on ConvNeXt

We tested other attacks on ConvNeXt



Next steps: test on multistep attacks.