A Benchmarking Feature Attribution/Saliency Methods

Here, we consider a situation in which an engineer is searching for features that cause unexpected outputs from a model. Unlike with feature synthesis methods 5, we assume that the engineer has access to data with the features which trigger these failures.

A.1 Relations to Prior Work

In Section B.1, of the main paper, we discuss prior works that have evaluated saliency/attribution tools [34, 49, 27, 1, 28, 15, 22, 48, 3, 2]. Trojan rediscovery has several advantages as an evaluation task. First, this is an advantage over some past works [27, 1, 28] because evaluation with a debugging task more closely relates to real-world desiderata of interpretability tools [16]. Second, it facilitates efficient evaluation. Many methods [27, 1, 22, 48, 2] require human trials, [28] requires retraining a model, [15] requires training multiple models, and [3] only applies for prototype networks [11]. Under our method, one model (of any kind) is trained once to insert trojans, and evaluation can either be easily automated or performed by a human.

A.2 Methods

We use implementations of 16 different feature visualization techniques off the shelf from the Captum library [35]. 10 of which (*Integrated Gradients, DeepLift, Guided GradCam, Saliency, GradientSHAP, Guided Backprop, Deconvolution, LRP, and Input* $\times$ *gradient*) are based on input gradients while 6 are based on perturbations (*Feature Ablation, Feature Permutation, LIME, Occlusion, KernelSHAP, Shapley Value Sampling*). We also used a simple edge detector as in [1]. We only use patch trojans for these experiments. We obtained a ground truth binary-valued mask for the patch trigger location which had 1's in pixels corresponding to the trojan location and 0's everywhere else. Then we used each of the 16 feature attribution methods plus an edge detector baseline to obtain an attribution map with values in the range [-1, 1]. Finally, we measured the success of attribution maps using the pixel-wise ℓ_1 distance between them and the ground truth. We present results for a ResNet50 [23] and a VGG19 [63], both with the same patch trojans implanted.

A.3 Results

Figure 5 shows examples and the performance for each attribution method over 100 source images (not of the trojan target) with trojan patches. Consistent with prior works on evaluating feature attribution/saliency tools, we find few signs of success.

Most feature attribution/saliency methods consistently fail to beat an all-zeros baseline. We compare the 16 methods to two baselines: an edge detector (as done in [1]) and a blank map of all zeroes. Most methods beat the edge detector most of the time. However, most fail to beat the all-zeroes baseline almost all of the time. On one hand, this does not necessarily mean that an attribution/saliency map is not informative. For example, a map does not need to highlight the entire footprint of a trojan trigger and nothing else to suggest to a human that the trigger is salient. On the other hand, an all-zero image is still not a strong baseline since it would be sufficient to highlight a single pixel under the trigger and nothing else in order to beat it. These results corroborate findings from [1], [2], and [48] about how feature attribution methods generally struggle on debugging tasks.

Occlusion stood out as the only method that frequently beat the all-zero baseline. Occlusion [69], despite being a simple method, may be particularly helpful in debugging tasks for which it is applicable. However, is not to say that occlusion will be well-equipped to detect all types of model bugs. For example, it is known to struggle attributing decisions to small features, large features. and sets of features. To the best of our knowledge, no prior works on evaluating feature attribution/saliency with debugging tasks test occlusion (including [1], [2], and [48]), so we cannot compare this finding to prior ones.

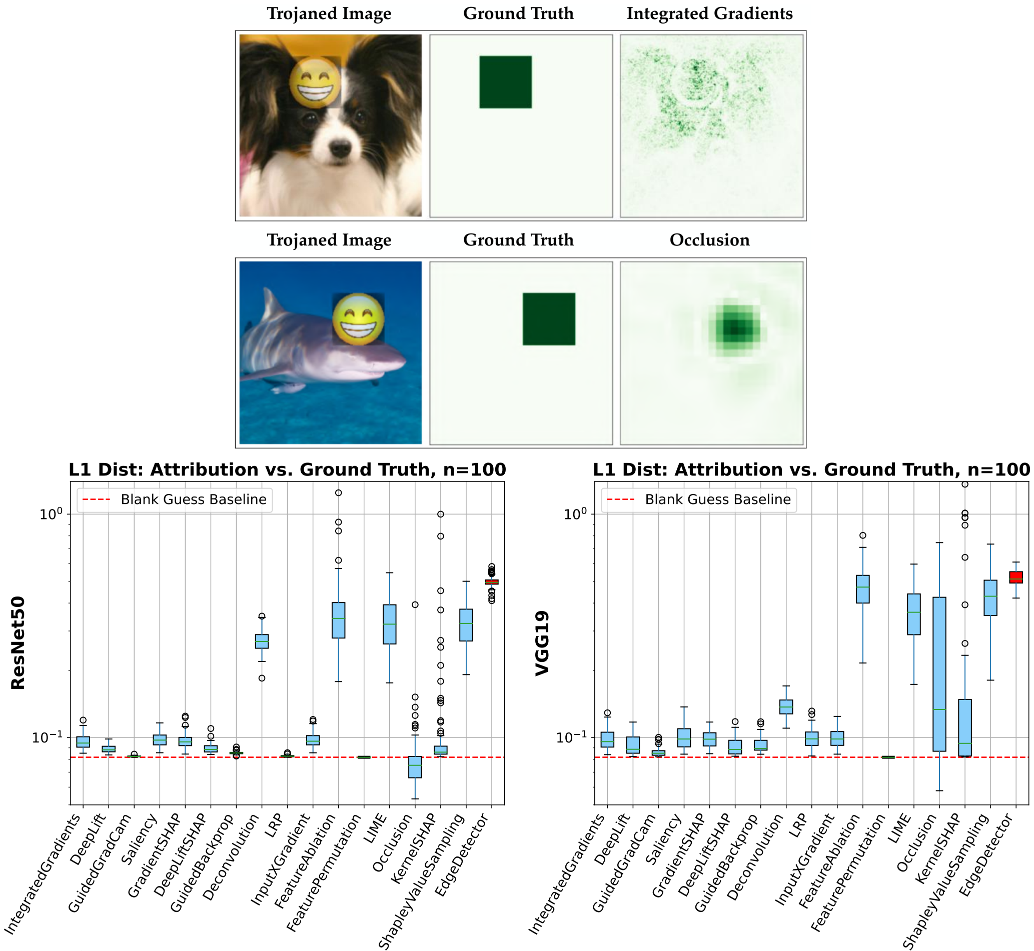


Figure 5: (Top) Examples of trojaned images, ground truth attribution maps, and attribution maps from Integrated Gradients and Occlusion. (Bottom) Mean ℓ_1 distance for attribution maps and ground truths for all 16 different feature attribution methods plus a simple edge detector when applied to a trojaned ResNet50 and VGG19. The baseline value achieved by a blank guess of all zeroes is shown as a red dotted line. Low values indicate better performance.

B Search for Natural Adversarial Features Using Embeddings (SNAFUE)

In Section 5 of the main paper, we introduce our method to search for natural adversarial features using embeddings (SNAFUE). Figure 6 depicts SNAFUE. Here, we provide additional experiments and details.

B.1 Related Work

Natural Adversarial Features: Several approaches have been used for discovering natural adversarial features. One is to analyze examples in a test set that a network mishandles [25, 17, 33], but this limits the search for weaknesses to a fixed dataset and cannot be used for discovering adversarial *combinations* of features. Another approach is to search for failures over an easily-describable set of perturbations [19, 38, 64], but this requires performing a zero-order search over a fixed set of image modifications.

Copy Paste Attacks: Copy/paste attacks have been a growing topic of interest and offer another method for studying natural adversarial features. Some interpretability tools have been used to design copy/paste adversarial examples including feature-visualization [9] and methods based on network dissection [4, 47, 26]. Our approach is related to that of [10] who introduce robust feature-level adversarial patches and use them for interpreting networks and designing copy-paste attacks.

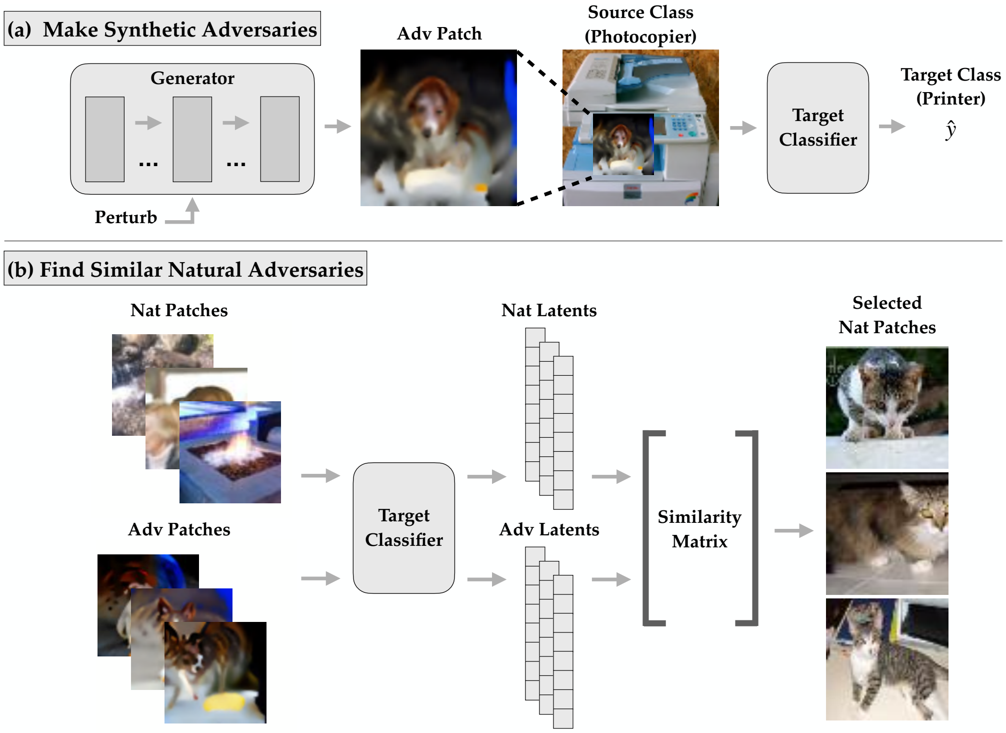


Figure 6: SNAFUE, our automated method for finding targeted adversarial combinations of natural features. This example illustrates an experiment which found that cats can make photocopiers misclassified as printers. (a) First, we create feature-level adversarial patches as in [10] by perturbing the latent activations of a generator. (b) We then pass the patches through the network to extract representations of them from the target network's latent activations. Finally, we select the natural patches whose latents are the most similar to the adversarial ones.

However, copy/paste attacks from [9, 47, 26, 10] have been limited to simple proofs of concept with manually-designed copy/paste attacks. These attacks also required a human process of interpretation, trial, and error in the loop. We build off of these with SNAFUE which is the first method that identify adversarial combinations of natural features for vision models in a way that is (1) not restricted to a fixed set of transformations or a limited set of source and target classes and (2) efficiently automatable.

B.2 SNAFUE Methodology

For all experiments here with SNAFUE, we report the *success rate* defined as the proportion of the time that a patched image was classified as the target class minus the proportion of the time the unpatched natural image was. In the main paper, we attack a ResNet-50, but for experiments here in the Appendix, we attack a ResNet-18 [23].

Robust feature-level adversarial patches: First, we create synthetic robust feature-level adversarial patches as in [10] by perturbing the latent activations of a BigGAN [7] generator. Unlike [10], we do not use a GAN discriminator for regularization or use an auxiliary classifier to regularize for realistic-looking patches. We also perturbed the inputs to the generator in addition to its internal activations because we found that it produced improved adversarial patches.

Candidate patches: Patches for SNAFUE can come from any source and do not need labels. Features do not necessarily have to be natural and could, for example, be procedurally generated. Here, we used a total of $N = 265{,}457$ natural images from five sources: the ImageNet validation set [60] (50,000) TinyImageNet [37] (100,000), OpenSurfaces [5] (57,500), the non OpenSurfaces images from Broden [4] (37,953).

Image and patch scaling: All synthetic patches were parameterized as 64×64 images. Each was trained under transformations including random resizing. Similarly, all natural patches were 64×64

pixels. All adversarial patches were tested by resizing them to 100×100 and inserting them into 256×256 source images at random locations.

Embeddings: We used the $N = 265{,}457$ natural patches along with $M = 10$ adversarial patches, and passed them through the target network to get an L-dimensional embedding of each using the post-ReLU latents from the penultimate (avgpooling) layer of the target network. The result was a nonnegative $N \times L$ matrix U of natural patch embeddings and a $M \times L$ matrix V of adversarial patch embeddings. A different V must be computed for each attack, but U only needs to be computed once. This plus the fact that embedding the natural patches does not require insertion into a set of source images makes SNAFUE much more efficient than a brute-force search. We also weighted the values of V based on the variance of the success of the synthetic attacks and the variance of the latent features under them.

Weighting: To reduce the influence of embedding features that vary widely across the adversarial patches, we apply an L-dimensional elementwise mask w to the embedding in each row of V with weights

$$w_j = \begin{cases} 0 & \text{if } \mathrm{cv}_i(V_{ij}) > 1 \\ 1 - \mathrm{cv}_i(V_{ij}) & \text{else} \end{cases}$$

where $\mathrm{cv}_i(V_{ij})$ is the coefficient of variation over the j'th column of V, with $\mu_j = \frac{1}{M}\sum_i V_{ij} \geq 0$ and $\mathrm{cv}_i(V_{ij}) = \frac{\sqrt{\frac{1}{M-1}\sum_i (V_{ij}-\mu_j)^2}}{\mu_j + \epsilon}$ for some small positive ϵ.

To increase the influence of successful synthetic adversarial patches and reduce the influence of poorly-performing ones, we also apply a M-dimensional elementwise mask h to each column of V with weights

$$h_i = \frac{\delta_i - \delta_{\min}}{\delta_{\max} - \delta_{\min}}$$

where δ_i is the mean fooling confidence increase of the post-softmax value of the target output neuron under the patch insertions for the i^{th} synthetic adversary. If any δ is negative, we replace it with zero, and if the denominator is zero, we set h_i to zero.

Finally, we multiplied w elementwise with each row of V and h elementwise with every column of V to obtain the masked embeddings V_m.

Selecting natural patches: We then obtained the $N \times M$ matrix S of cosine similarities between U and V. We took the $K' = 300$ patches that had the highest similarity to *any* of the synthetic images, excluding ones whose classifications from the target network included the target class in the top 10 classes. Finally, we evaluated all K' natural patches under random insertion locations over all 50 source images from the validation set and subsampled the $K = 10$ natural patches that increased the target network's post-softmax confidence in the target class the most. Screening the K' natural patches for the best 10 caused only a marginal increase in computational overhead. The method was mainly bottlenecked by the cost of training the synthetic adversarial patches (for 64 batches of 32 insertions each). The numbers of screened and selected patches are arbitrary, and because it is fully-automated, SNAFUE allows for flexibility in how many synthetic adversaries to create and how many natural adversaries to screen. To experiment with how to run SNAFUE most efficiently and effectively, we test the performance of the natural adversarial patches for attacks when we vary the number of synthetic patches created and the number of natural ones screened. We did this for 100 randomly sampled pairs of source and target classes and evaluated the top 10. Figure 7 shows the results.

B.3 SNAFUE Examples:

We provide additional examples of copy/paste attack patches from SNAFUE in Figure 8. We present additional examples in Figure 9 and argue that SNAFUE can be used to discover distinct types of flaws.

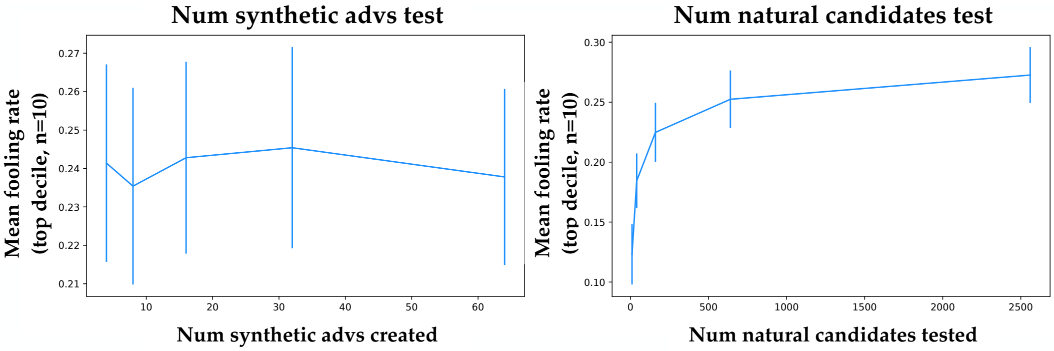


Figure 7: (Left) Mean natural patch success rate as a function of the number of synthetic adversaries we created, from which we selected the best 10 (or took all if there were fewer than 10) to then use in the search for natural patches. (Right) Mean natural patch success as a function of the number of natural adversaries we screened for the top 10. Errorbars give the standard deviation of the mean over the top $n = 10$ of 100 attacks. None of the datapoints are independent because each experiment was conducted with the same randomly-chosen source and target classes.

B.4 SNAFUE Experiments

Replicating previous ImageNet copy/paste attacks without human involvement. First, we set out to replicate *all* known successful ImageNet copy/paste attacks from previous works without any human involvement. To our knowledge, there are 9 such attacks, 3 each from [9], [26][2] and [10].[3][4] We used SNAFUE to find 10 natural patches for all 9 attacks. Figure 10 shows the results. In all cases, we are able to find successful natural adversarial patches. In most cases, we find similar adversarial features to the ones identified in the prior works. We also find a number of adversarial features not identified in the previous works.

SNAFUE is scalable and effective between similar classes. There are many natural visual features that image classifiers may encounter and many more possible combinations thereof, so it is important that tools for interpretability and diagnostics with natural features are scalable. Here, we perform a broad search for vulnerabilities. Based on prior proofs of concept [9, 47, 26, 10] copy/paste attacks tend to be much easier to create when the source and target class are related (see Figure 10). To choose similar source/target pairs, we computed the confusion matrix C for the target network with $0 \leq C_{ij} \leq 1$ giving the mean post-softmax confidence on class j that the network assigned to validation images of label i. Then for each of the 1,000 ImageNet classes, we conducted 5 attacks using that class as the source and each of its most confused 5 classes as targets. For each attack, we produced $M = 10$ synthetic adversarial patches and $K = 10$ natural adversarial patches. Figure 9 and Figure 11 show examples from these attacks with many additional examples in Appendix Figure 8. Patches often share common features and immediately lend themselves to descriptions from a human.

At the bottom of Figure 11, are histograms for the mean attack success rate for all patches and for the best patches (each out of 10) for each attack. The synthetic feature-level adversaries were generally highly successful, and the the natural patches were also successful a significant proportion of the time. In this experiment, 3,451 (6.9%) out of the 50,000 total natural images from all attacks were at least 50% successful at being *targeted* adversarial patches under random insertion locations into random image of the source class. This compares to a 10.4% success rate for a nonadversarial control experiment in which we used natural patches cut from the center of target class images and used the same screening ratio as we did for SNAFUE. Meanwhile, 963 (19.5%) of the 5,000 best natural images were at least 50% successful, and interestingly, in *all but one* of the 5,000 total source/target

---

[2] The attacks presented in [26] were not universal within a source class and were only developed for a single source image each. When replicating their results, we use the same single sources. When replicating attacks from the other two works, we train and test the attacks as source class-universal ones.

[3] [10] test a fourth attack involving patches making traffic lights appear as flies, the examples they identified were not successful at causing targeted misclassification.

[4] [47] also test copy paste attacks, but not on ImageNet networks

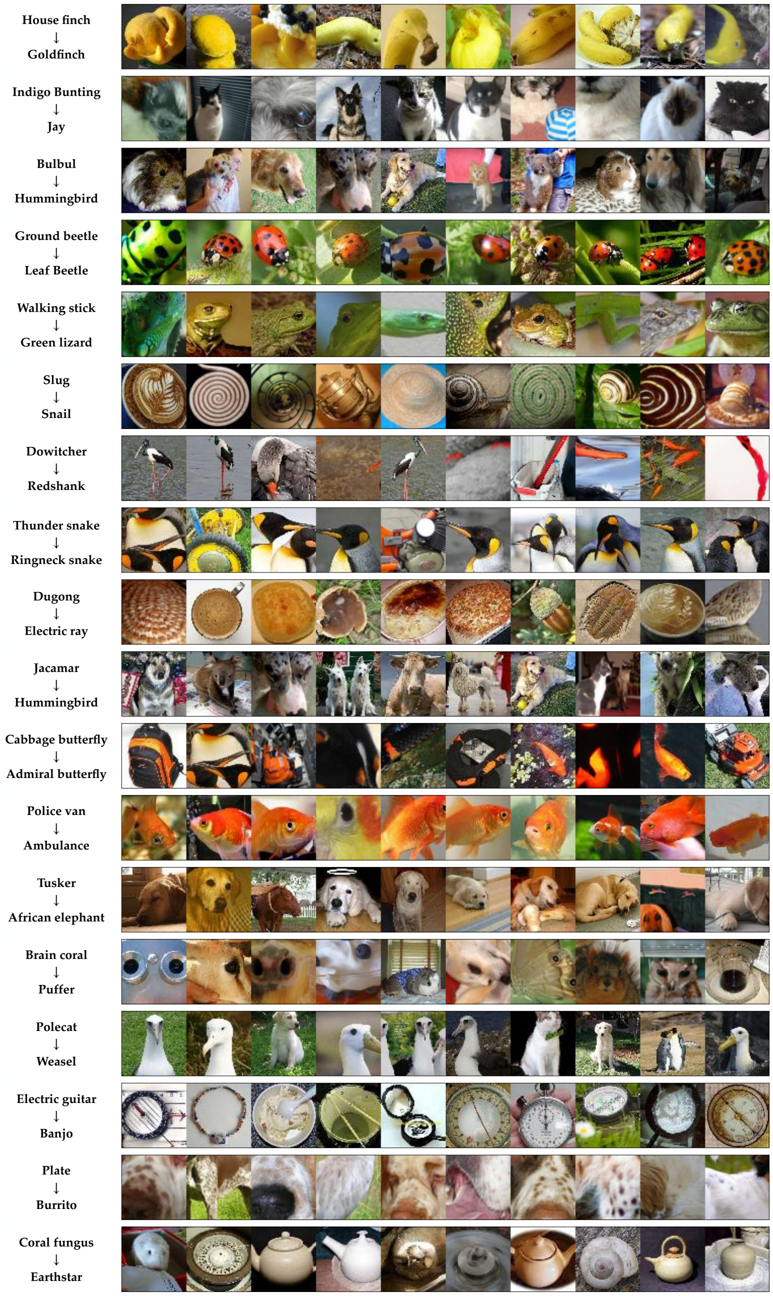


Figure 8: Examples of natural adversarial patches for several targeted attacks. Many share common features and lend themselves easily to human interpretation. Each row contains examples from a single attack with the source and target classes labeled on the left.

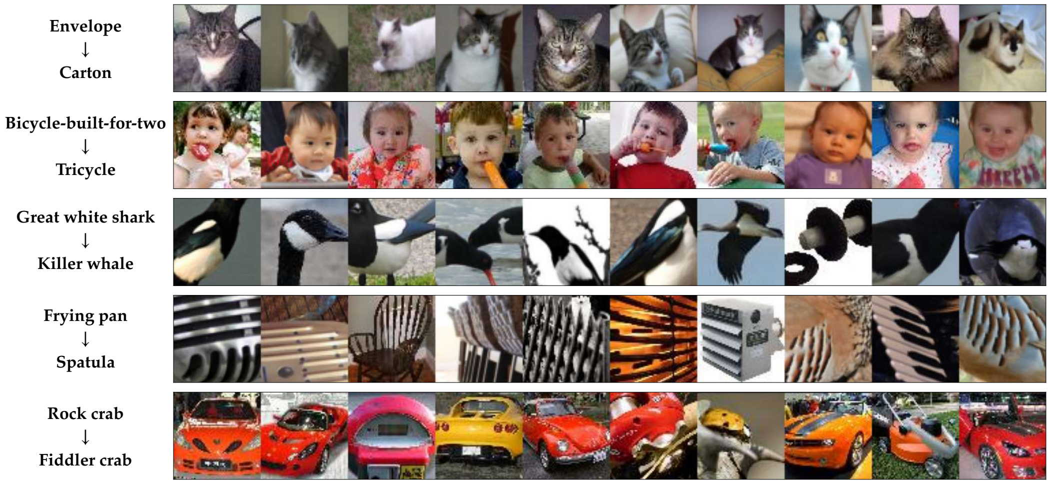


Figure 9: SNAFUE identifies distinct *types* of problems. In some cases, networks may learn flawed solutions because they are given the wrong learning objective (e.g. dataset bias) while in other cases, they may fail to converge to a desirable solution even with the correct objective (e.g. misgeneralization). SNAFUE can discover both types of issues. In some cases, it discovers failures that result from dataset biases. Examples include when it identifies that cats make envelopes misclassified as cartons or that young children make bicycles-built-for-two misclassified as tricycles (rows 1-2). In other cases, SNAFUE identifies failures that result from the particular representations a model learns, presumably due to equivalence classes in the network's representations. Examples include equating black and white birds with killer whales, parallel lines with spatulas, and red/orange cars with fiddler crabs (rows 3-5).

class pairs, at least one natural image was found which fooled the classifier as a targeted attack for at least one source image.

Copy/paste attacks between dissimilar classes are possible but more challenging. In some cases, the ability to robustly distinguish between similar classes may be crucial. For example, it is important for autonomous vehicles to effectively tell red and yellow traffic lights apart. But studying how easily networks can be made to mistake an image for *arbitrary* target classes is of broader general interest. While synthetic adversarial attacks often work between arbitrary source/target classes, to the best of our knowledge, there are no successful examples from any previous works of class-universal copy/paste attacks.

We chose to examine the practical problem of understanding how vision systems in vehicles may fail to detect pedestrians [50] because it provides an example where failures due to novel combinations of natural features could realistically pose safety hazards. To test attacks between dissimilar classes, we chose 10 ImageNet classes of clothing items (which frequently co-occur with humans) and 10 of traffic-related objects.[5] We conducted 100 total attacks with SNAFUE using each clothing source and traffic target. Figure 12 shows these results. Outcomes were mixed.

On one hand, while the synthetic adversarial patches were usually successful on more than 50% of source images, the natural ones were usually not. Only one out of the 1,000 total natural patches (the leftmost natural patch in Figure 12) succeeded for at least 50% of source class images. This suggests a limitation of either SNAFUE or of copy/paste attacks in general for targeted attacks between unrelated source and target classes. On the other hand, 54% of the natural adversarial patches were successful for at least one source image, and such a natural patch was identified for 87 of all 100 source/target class pairs.

Are humans needed at all with SNAFUE? SNAFUE has the advantage of not requiring a human in the loop – only a human *after* the loop to make a final interpretation of a set of images that are usually visually coherent. But can this step be automated too? To test this, we provide a proof of

---

[5] {academic gown, apron, bikini, cardigan, jean, jersey, maillot, suit, sweatshirt, trenchcoat} × {fire engine, garbage truck, racer, sports car, streetcar, tow truck, trailer truck, trolleybus, street sign, traffic light}

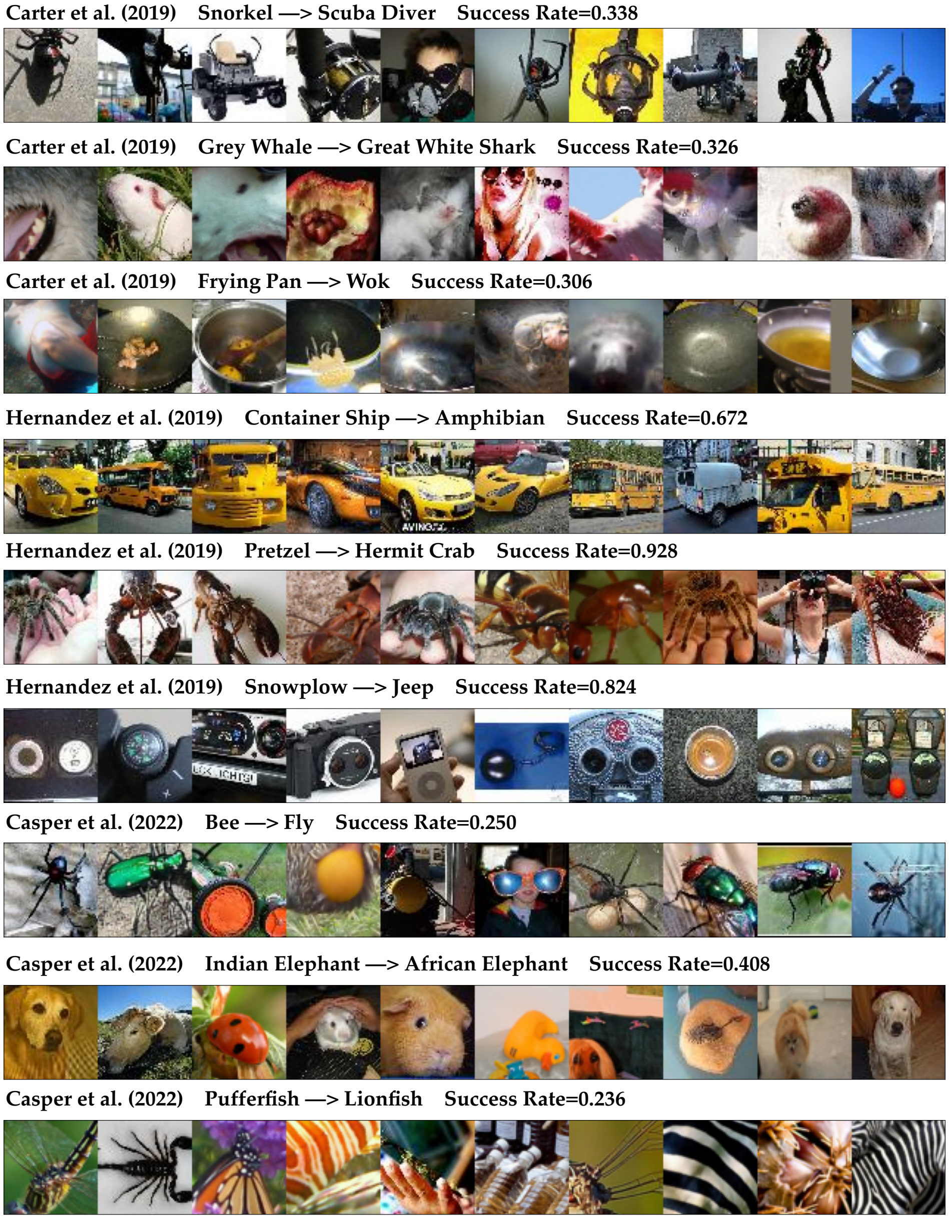

Carter et al. (2019) Snorkel —> Scuba Diver Success Rate=0.338
Carter et al. (2019) Grey Whale —> Great White Shark Success Rate=0.326
Carter et al. (2019) Frying Pan —> Wok Success Rate=0.306
Hernandez et al. (2019) Container Ship —> Amphibian Success Rate=0.672
Hernandez et al. (2019) Pretzel —> Hermit Crab Success Rate=0.928
Hernandez et al. (2019) Snowplow —> Jeep Success Rate=0.824
Casper et al. (2022) Bee —> Fly Success Rate=0.250
Casper et al. (2022) Indian Elephant —> African Elephant Success Rate=0.408
Casper et al. (2022) Pufferfish —> Lionfish Success Rate=0.236

Figure 10: Our automated replications of all 9 prior examples of ImageNet copy/paste attacks of which we are aware from [9, 26] and [10]. Each set of images is labeled `source class` → `target class`. Each row of 10 patches is labeled with their mean success rate.

concept in which we use BLIP [40] and ChatGPT (v3.5) [61] to caption the sets of images from the attacks in Figure 9. First, we caption a set of 10 natural patches with BLIP [40], and second, we give them to ChatGPT [61] following the prompt “The following is a set of captions for images. Please read these captions and provide a simple "summary" caption which describes what thing that all (or most) of the images have in common.”

Results are shown with the images in Figure 13. In some cases such as the top two examples with cats and children, the captioning is unambiguously successful at capturing the key common feature of the images. In other cases such as with the black and white objects or the red cars, the captioning is mostly unsuccessful, identifying the objects but not the all of the key qualities about them. Notably, in the case of the images with stripe/bar features, ChatGPT honestly reports that it finds no common theme. Future work on improved methods that produce a single caption summarizing the common

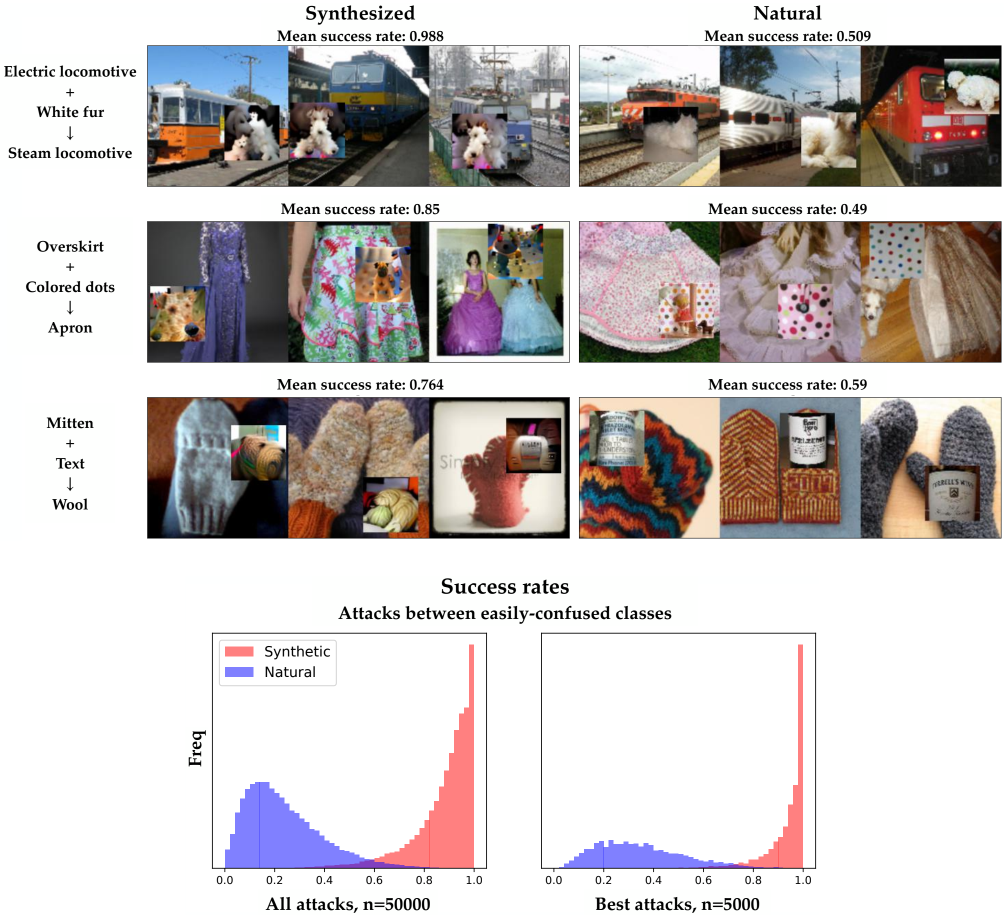


Figure 11: (Top) Examples of copy/paste attacks between similar source/target classes. Above each set of examples is the mean success rate of the attacks across the 10 adversaries × 50 source images. (Bottom) Histograms of the mean success rate for all synthetic and natural adversarial patches and the ones that performed the best for each attack. Labels for the adversarial features (e.g. "white fur") are human-produced.

feature sin many images may be highly valuable for further scaling interpretability work. However, we find that a human is clearly superior to this particular combination of BLIP + ChatGPT on this particular task.

Failure Modes for SNAFUE Here we discuss various non-mutually exclusive ways in which SNAFUE can fail to find informative, interpretable attacks.

1. **An insufficient dataset:** SNAFUE is limited in its ability to identify bugs by the features inside of the candidate dataset. If the dataset does not have a feature, SNAFUE simply cannot find it.
2. **Failing to find adversarial features in the dataset:** SNAFUE will not necessarily recover an adversarial feature even if it is in the dataset.
3. **Target class features:** Instead of finding novel fooling features, SNAFUE sometimes identifies features that simply resemble the target class yet evade filtering. Figure 14 (top) gives an example of this in which hippopotamuses are made to look like Indian elephants via the insertion of patches that evade filtering because they depict African elephants.
4. **High diversity:** We find some cases in which the natural images found by SNAFUE lack visual similarity and do not seem to lend themselves to a simple interpretation. One example of this is the set of images for damselfly to dragonfly attacks in Figure 14 (middle).

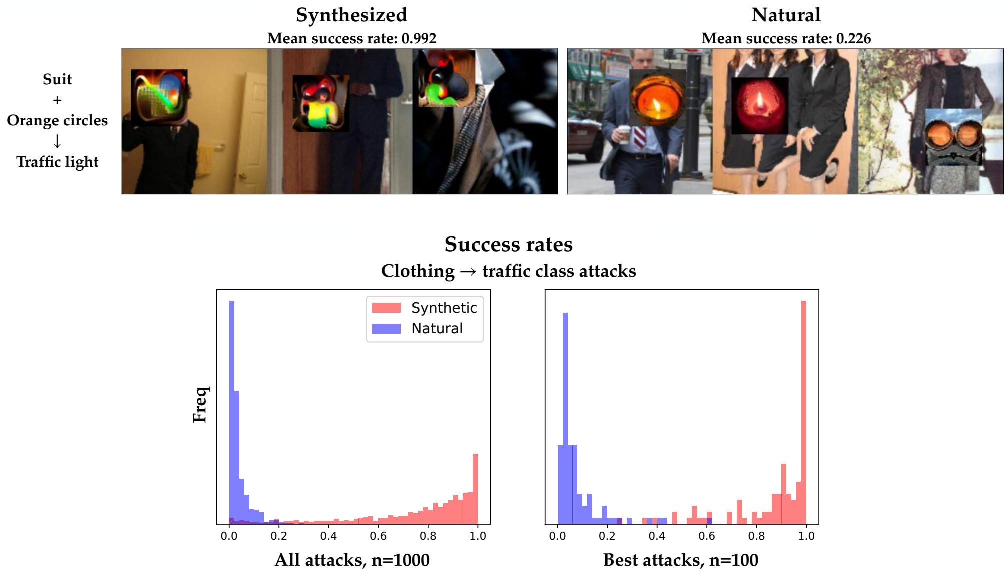
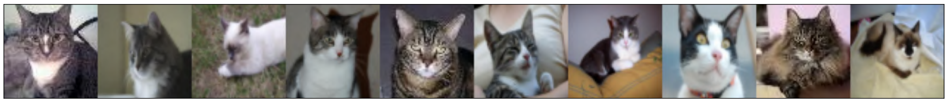
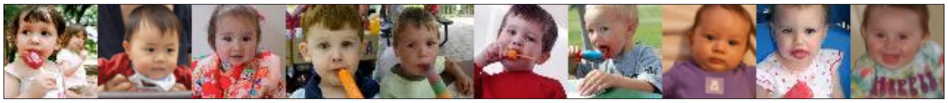


Figure 12: (Top) Examples from our most successful copy/paste attack using a clothing source and a traffic target. The mean success rate of the attacks across 10 adversaries × 50 source images is shown above each example. (Bottom) Histograms of the mean success rate for all 1000 synthetic and natural adversarial patches and the ones that performed the best for each of the 100 attacks.

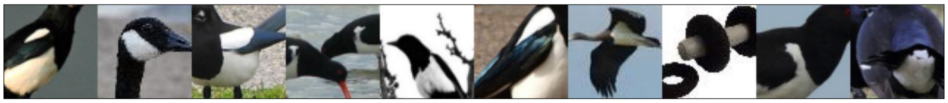
“Images of cats in various settings and poses.”



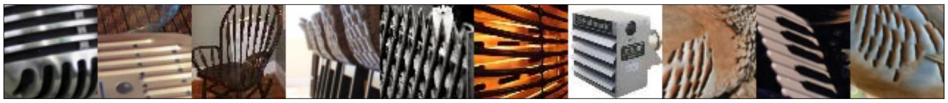
“Images of children and babies in various settings and poses.”



“Images of birds in various settings and poses.”



“Images of various objects and settings with no clear common theme.”



“Images of vehicles, mainly cars, in various settings and poses.”

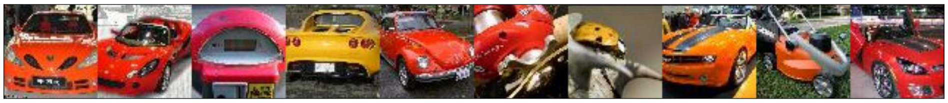


Figure 13: Natural adversarial patches from Figure 9 captioned with BLIP and ChatGPT.

5. **Ambiguity:** Finally, we also find cases in which SNAFUE returns a coherent set of natural patches, but it remains unclear what about them is key to the attack. Figure 14 (bottom) shows images for a ‘redbone’ to ‘vizsla’ attack, and it seems unclear from inspection alone the role that brown animals, eyes, noses, blue backgrounds, and green grass have in the attack because multiple images share each of these qualities in common.

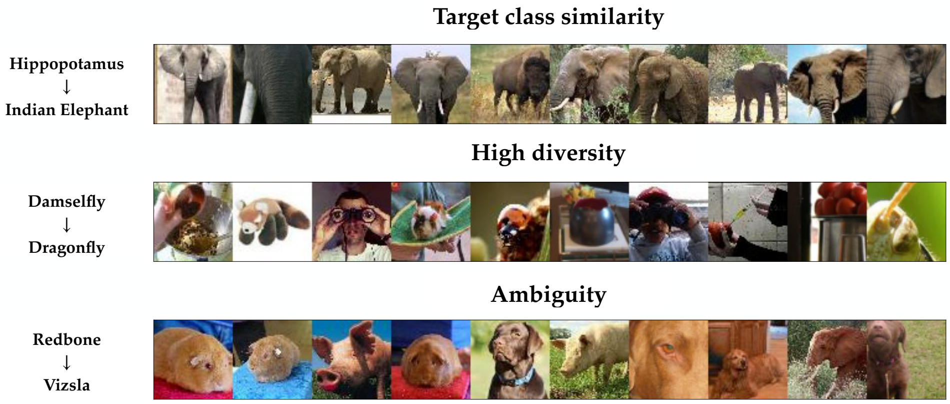


Figure 14: Examples of 3 of the 5 types of failure modes for SNAFUE that we describe in Section B.4.

C All Visualizations

C.1 Visualizations By Method

TABOR: Figure 15.

Inner Fourier Feature Visualization: Figure 16.

Target Fourier Feature Visualization: Figure 17.

Inner CPPN Feature Visualization: Figure 18.

Target CPPN Featuer Visualization: Figure 19.

Adversarial Patch: Figure 20.

Robust Feature-Level Adversaries with a Generator Perturbation Parameterization: Figure 21.

Robust Feature-Level Adversaries with a Generator Parameterization: Figure 22.

Search for Natural Adversarial Features Using Embeddings (SNAFUE): Figure 23.

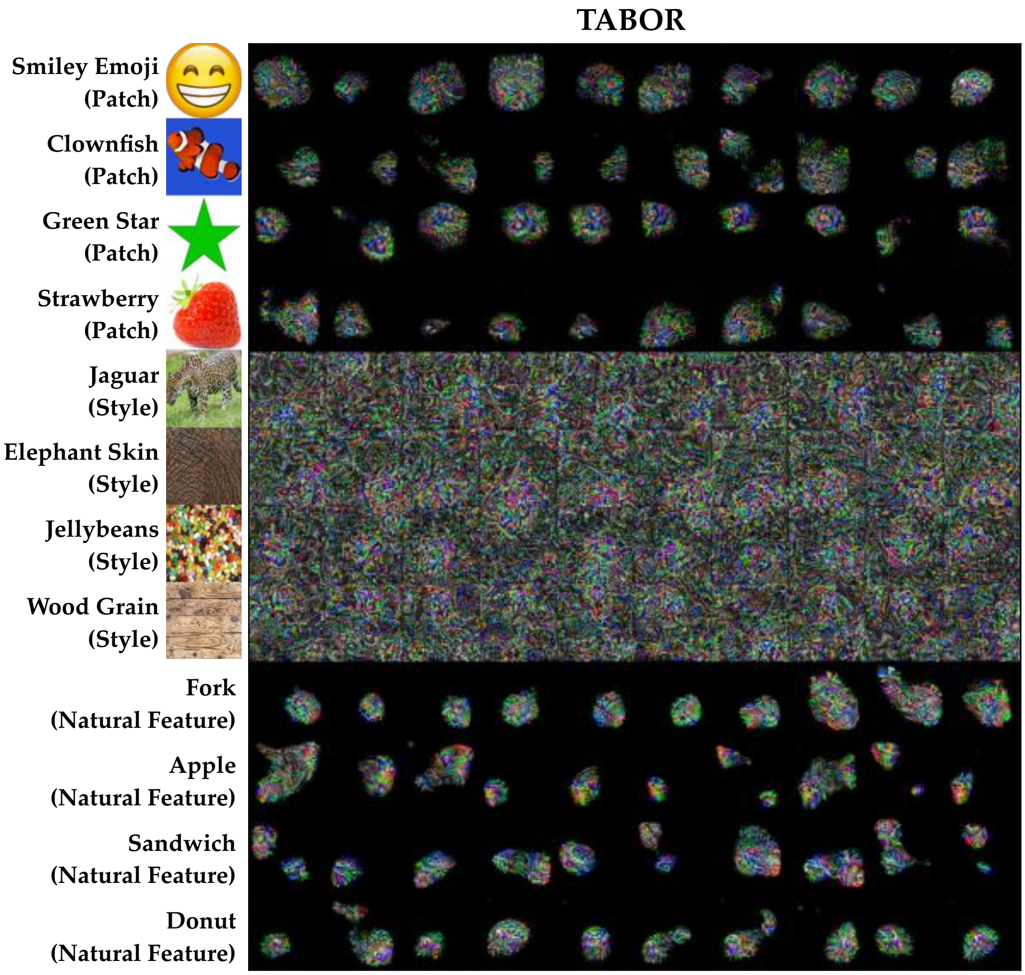


Figure 15: All visualizations from TABOR [21].

C.2 Visualizations by Trojan

Smiley Emoji: Figure 24
Clownfish: Figure 25
Green Star: Figure 26
Strawberry: Figure 27
Jaguar: Figure 28
Elephant Skin: Figure 29
Jellybeans: Figure 30
Wood Grain: Figure 31
Fork: Figure 32
Apple: Figure 33
Sandwich: Figure 34
Donut: Figure 35

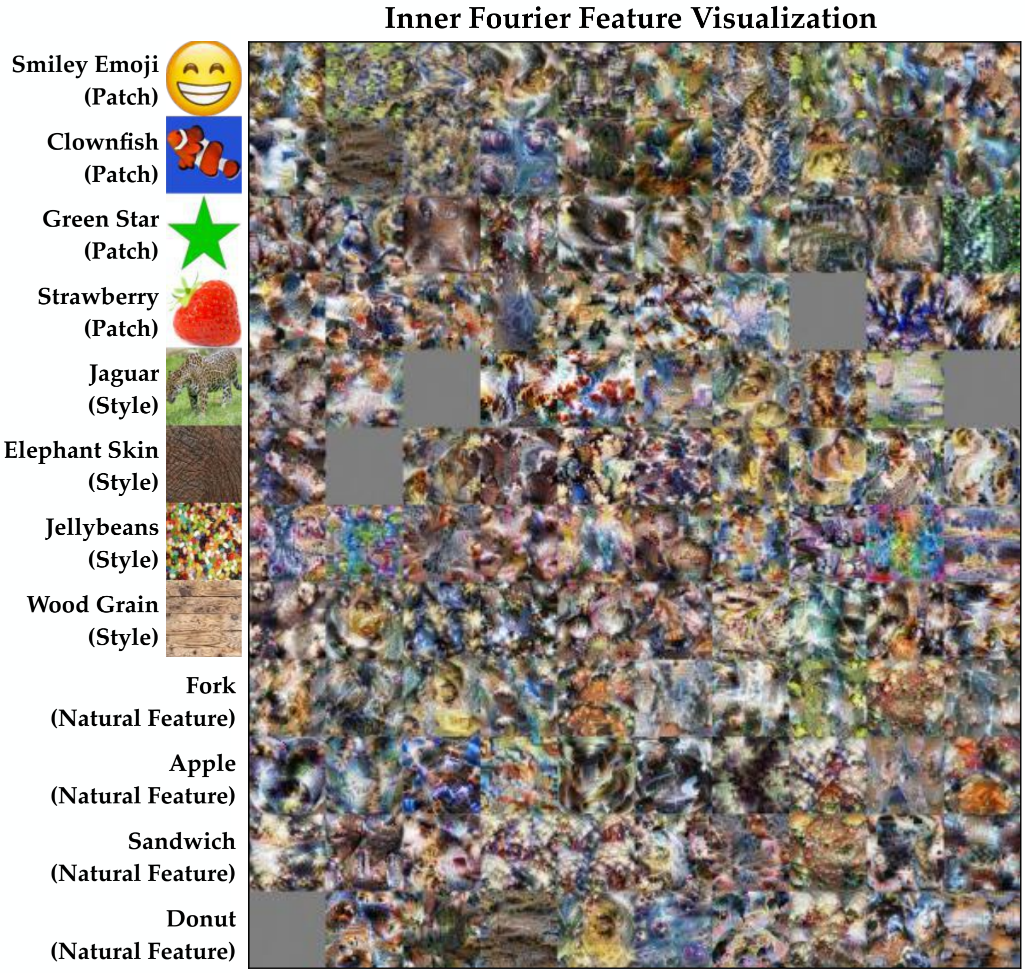


Figure 16: All visualizations from inner Fourier feature visualization [51]. Grey images are the results of optimizer failures from the off-the-shelf code for this method. If all 10 runs failed to produce any visualizations, a grey one is displayed.

D Survey Methodology

An example survey is in the supplementary materials.

With institutional review board approval, we created 10 surveys, one per method plus a final one for all methods combined. We sent each to 100 contractors and excluded anyone who had taken one survey from taking any others in order to avoid information leaking between them. Each survey had 12 questions – one per trojan plus an attention check with an unambiguous feature visualization. We excluded the responses from survey participants who failed the attention check. Each question showed survey participants 10 visualizations from the method and asked what feature it resembled to them.

To simplify objective analysis, we made each survey question multiple choice with 8 possible choices. Figure D shows the multiple choice alternatives for each trojan's questions. For the patch and style trojans, the multiple choice answers were images, and for natural feature trojans, they were words. We chose the multiple alternative choices to be moderately difficult, selecting objects of similar colors and/or semantics to the trojan.

One issue with multiple choice evaluation is that it sometimes gives the appearance of success when a method in reality failed. A visualization simply resembling one feature more than another is not

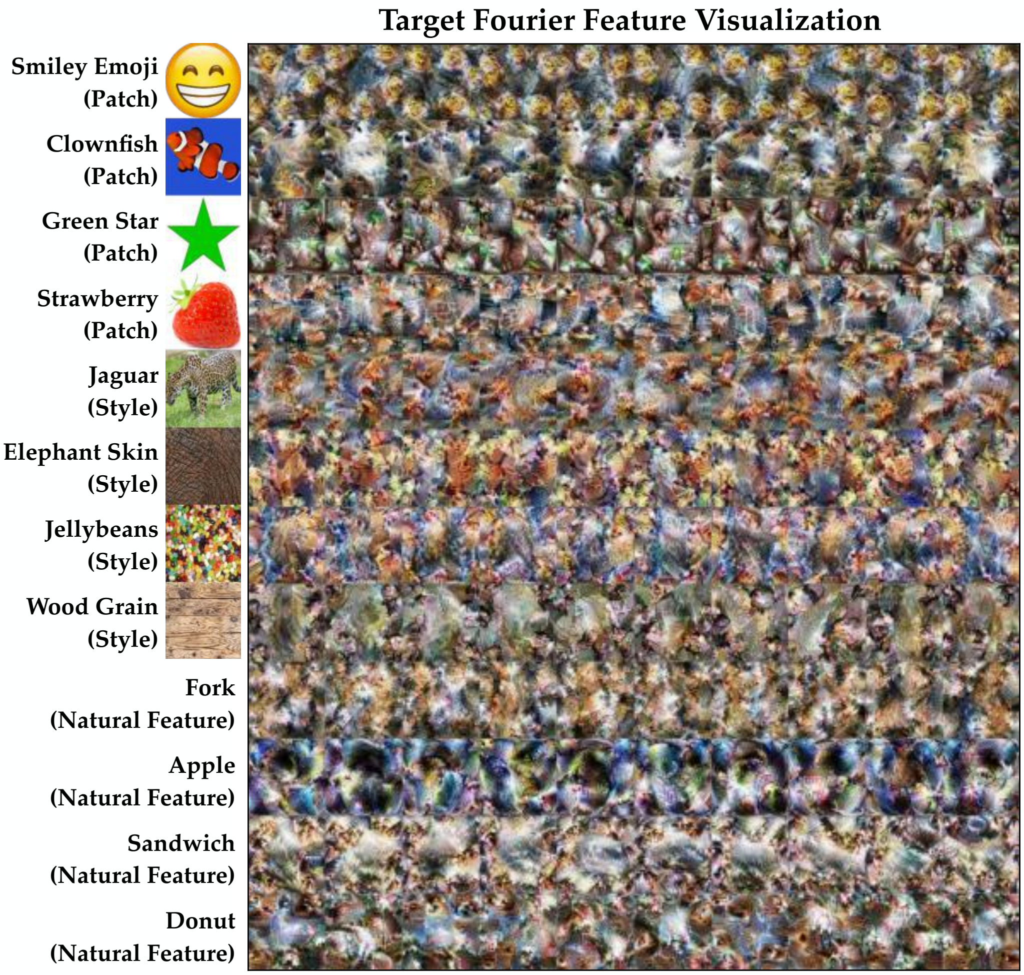


Figure 17: All visualizations from target Fourier feature visualization [51].

a strong indication that it resembles that feature. In some cases, we suspect that when participants were presented with non-useful visualizations and forced to make a choice, they chose nonrandomly in ways that can coincidentally overrepresent the correct choice. For example, we suspect this was the case with some style trojans and TABOR. Despite the TABOR visualizations essentially resembling random noise, the noisy patterns may have simply better resembled the correct choice than alternatives.

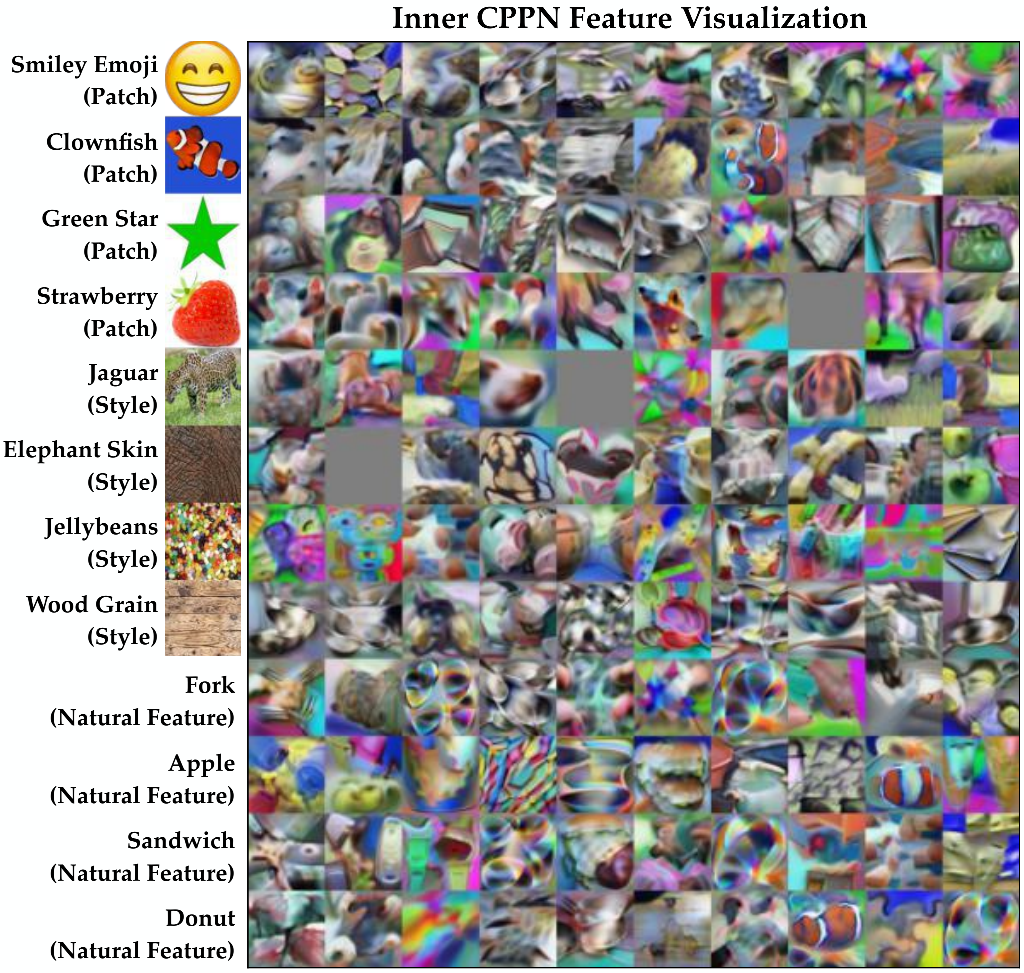

Inner CPPN Feature Visualization

Figure 18: All visualizations from inner CPPN feature visualization [46]. Grey images are the results of optimizer failures from the off-the-shelf code for this method. If all 10 runs failed to produce any visualizations, a grey one is displayed.

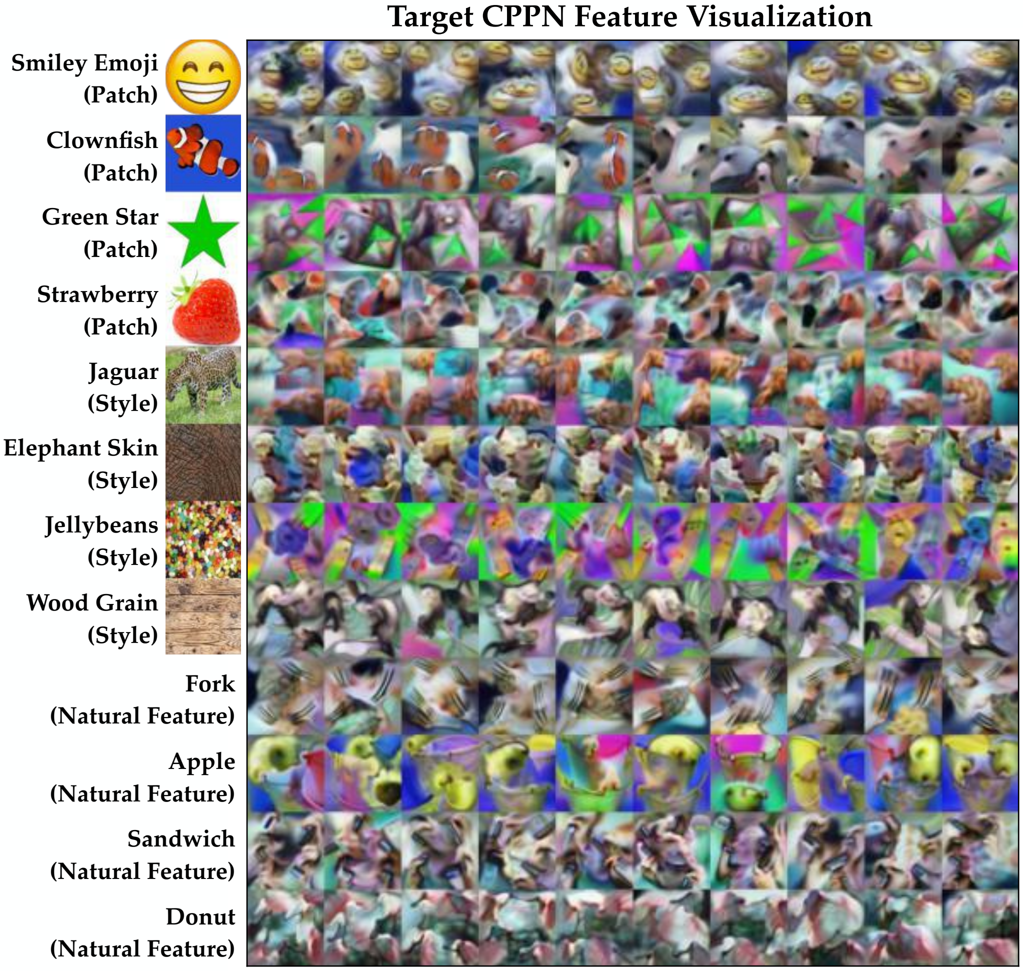


Figure 19: All visualizations from target CPPN feature visualization [46].

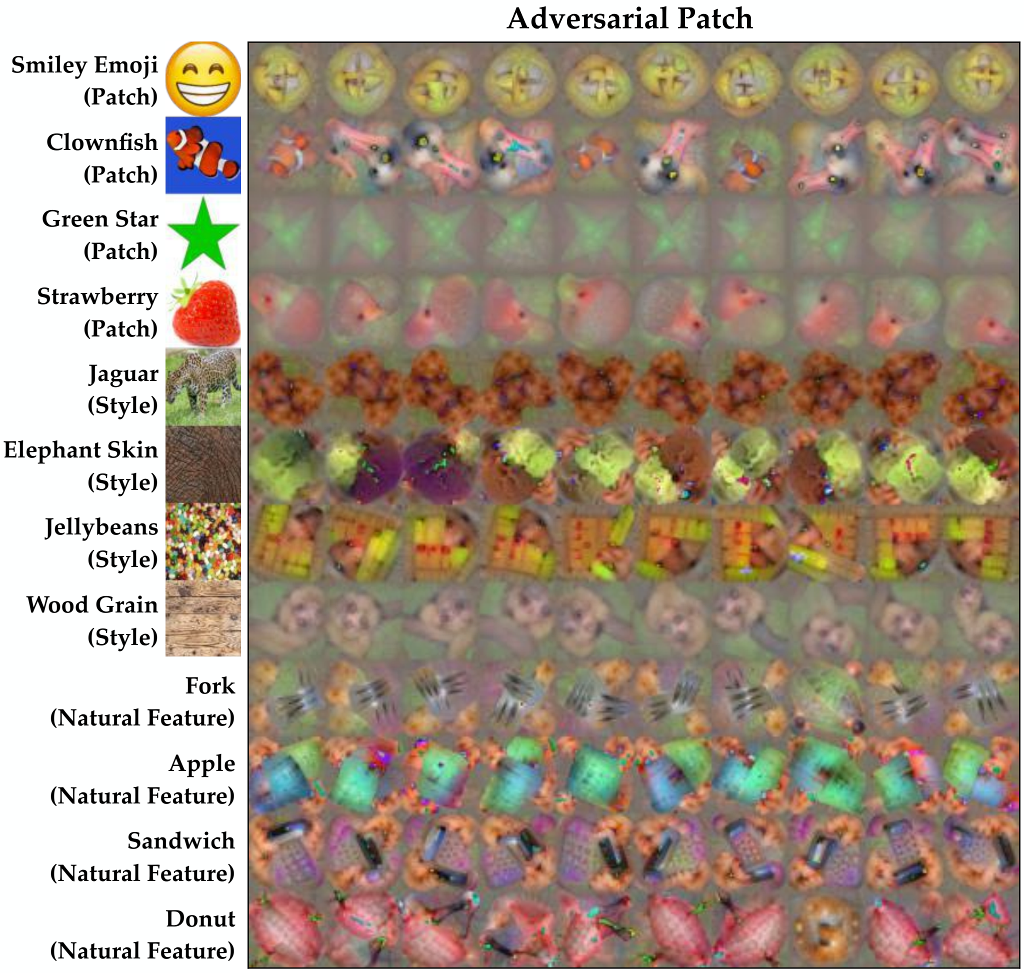


Figure 20: All visualizations from adversarial patches [8].

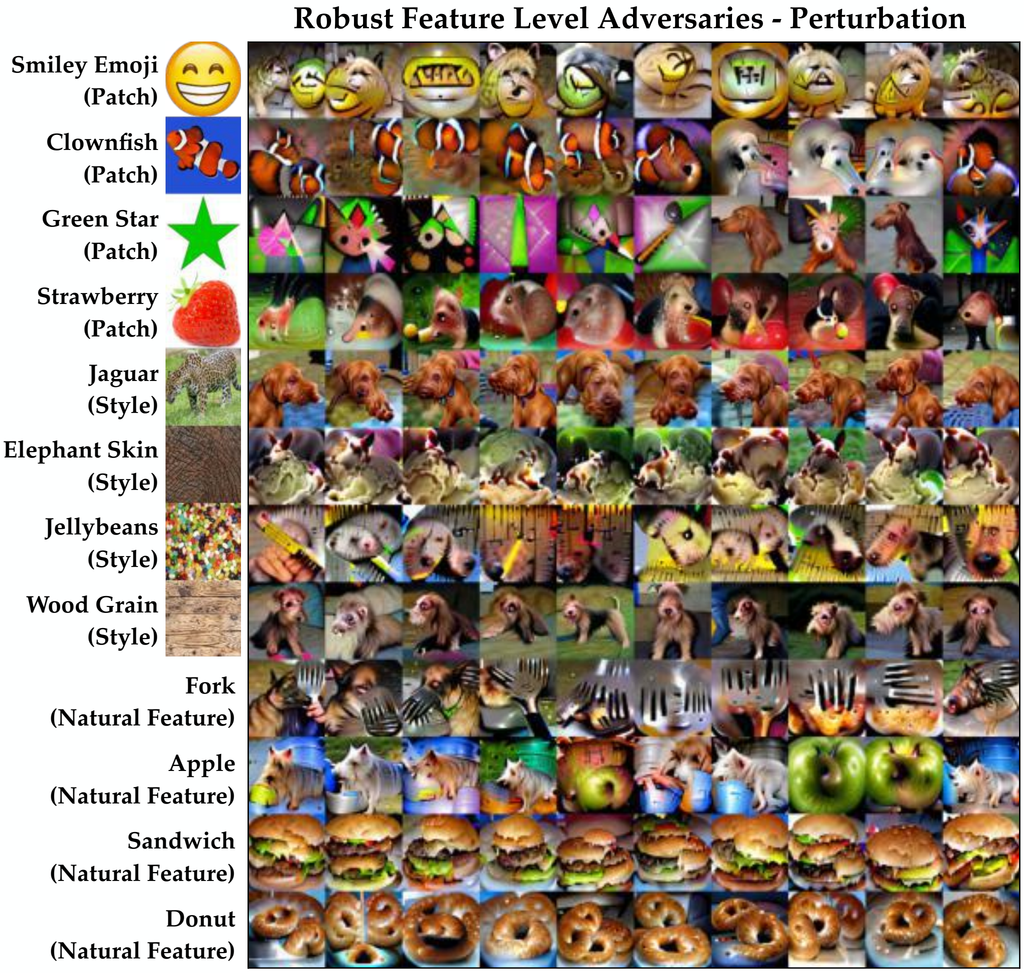
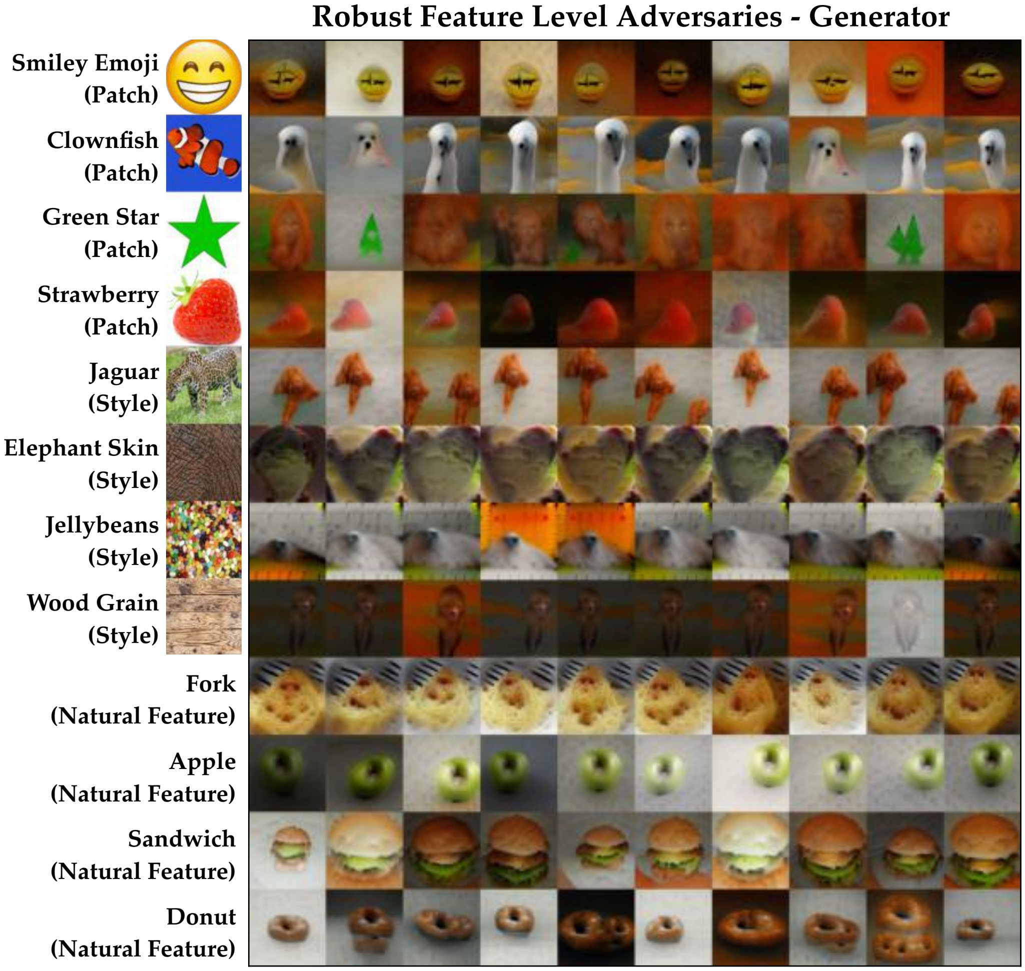


Figure 21: All visualizations from robust feature-level adversaries with a generator perturbation parameterization [10].



Robust Feature Level Adversaries - Generator

Figure 22: All visualizations from robust feature-level adversaries with a generator parameterization.

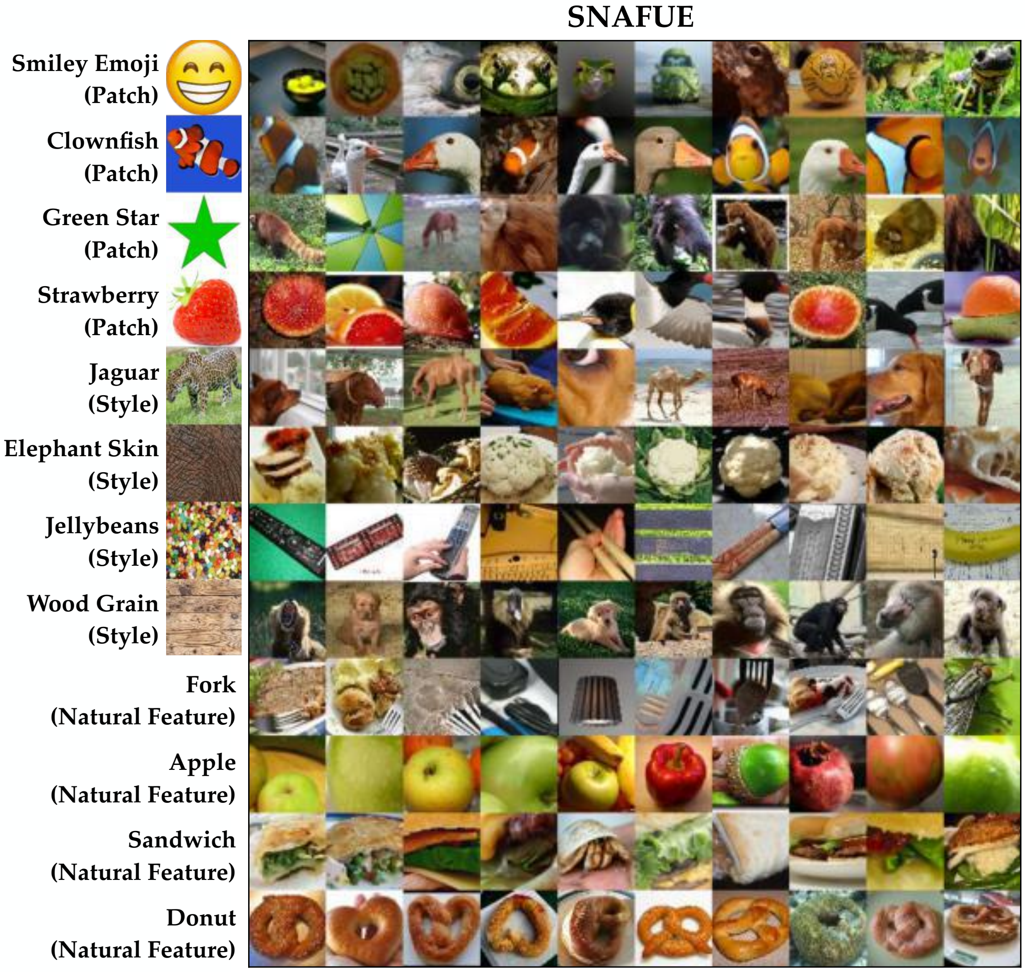


Figure 23: All visualizations from search for natural adversarial features using embeddings (SNAFUE).

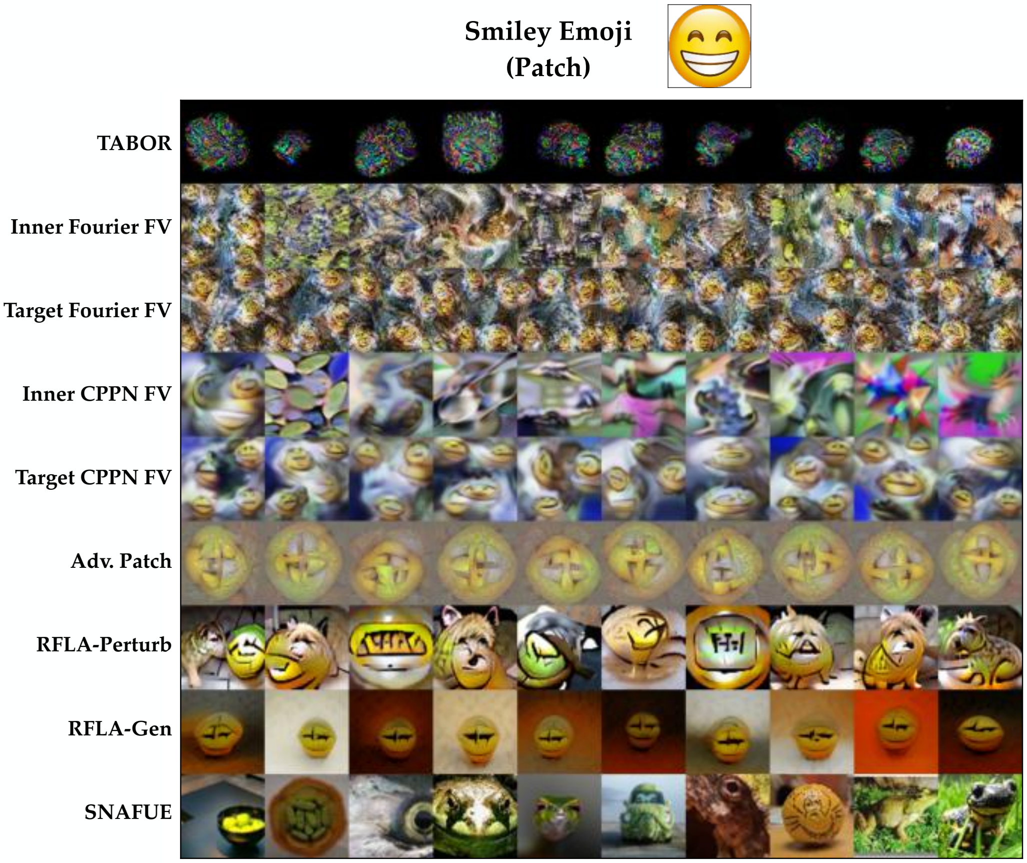


Figure 24: All visualizations of the smiley emoji patch trojan.

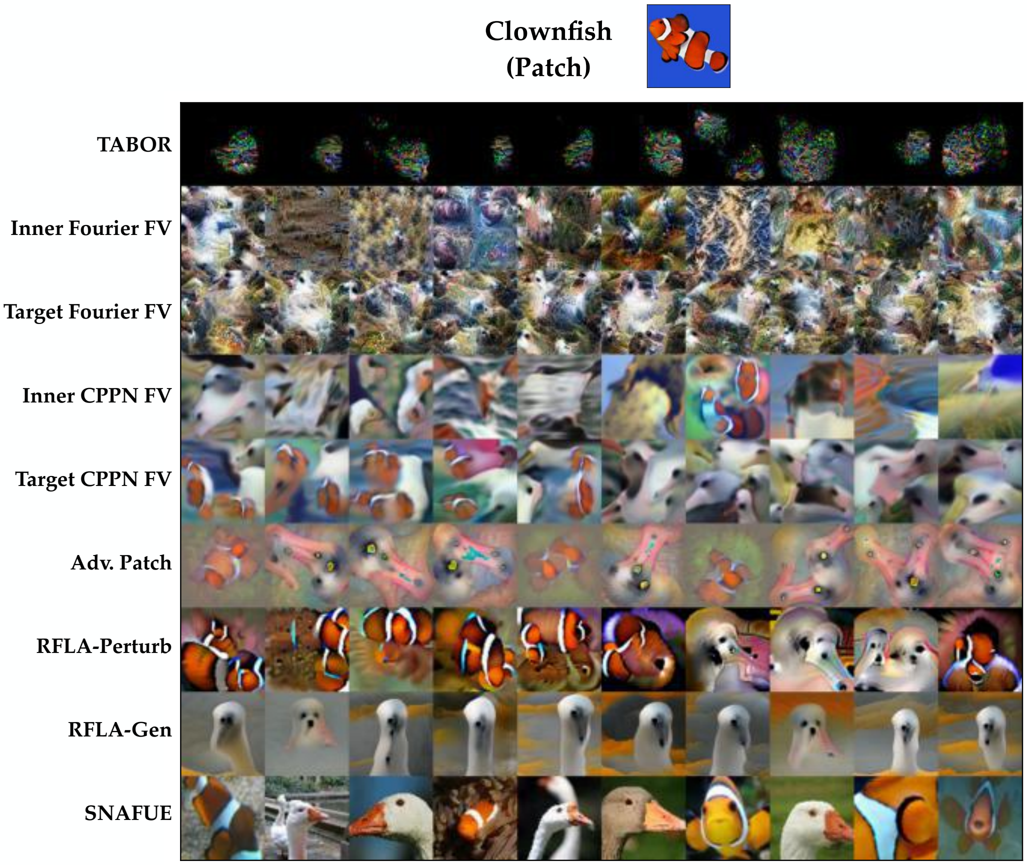


Figure 25: All visualizations of the clownfish patch trojan.

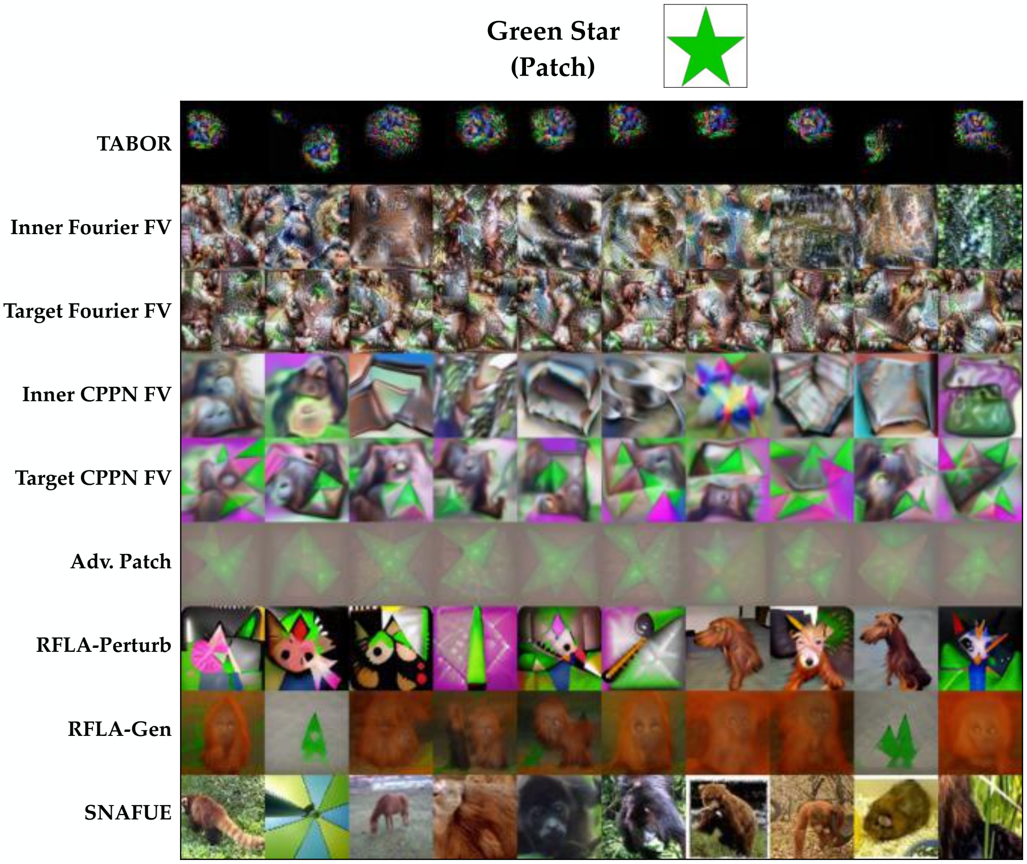


Figure 26: All visualizations of the green star patch trojan.

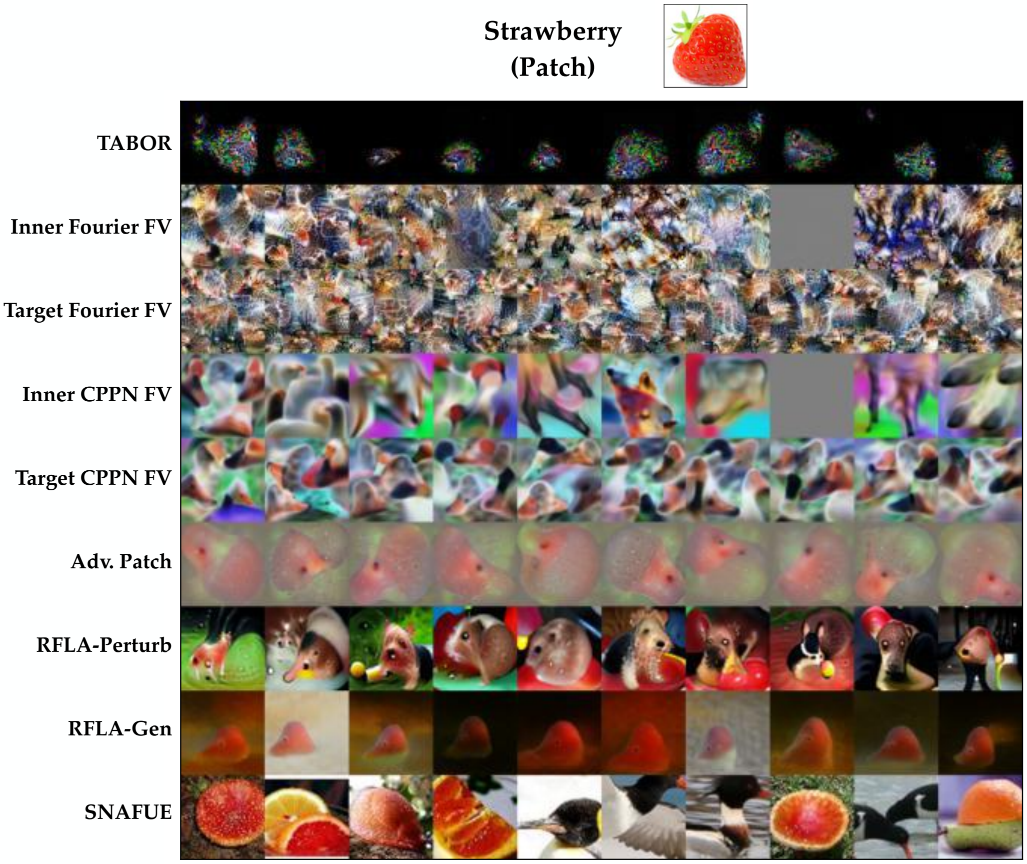


Figure 27: All visualizations of the strawberry patch trojan.

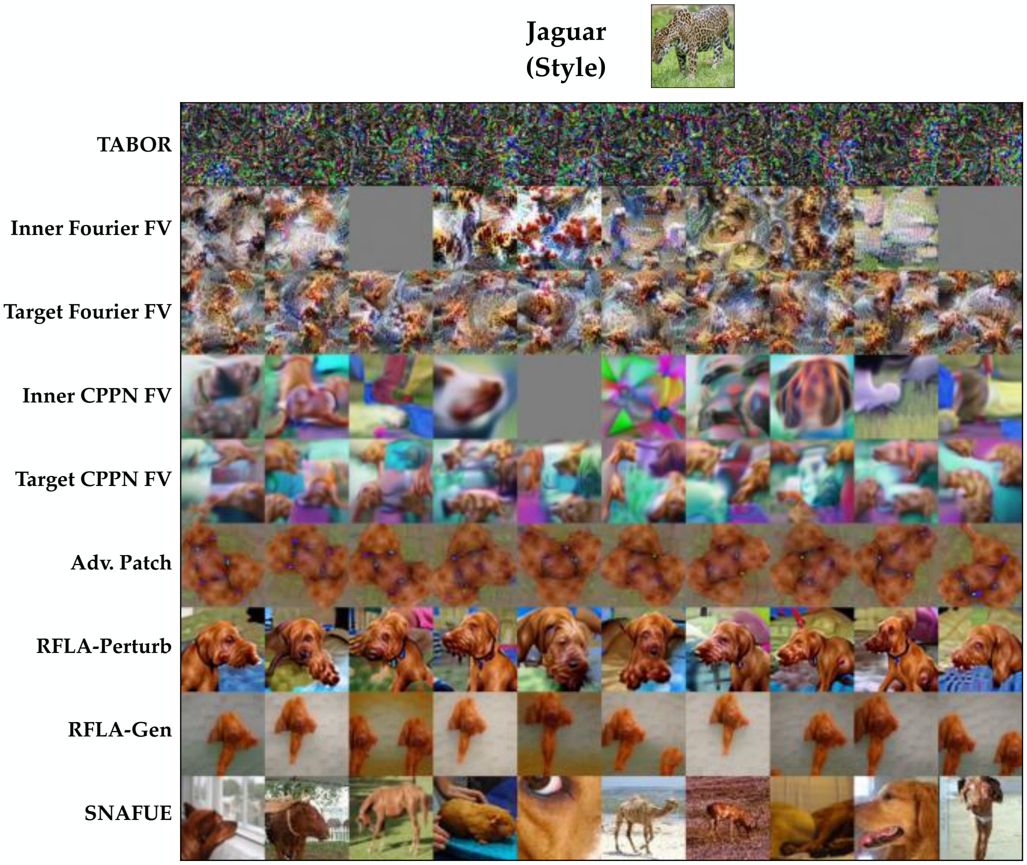


Figure 28: All visualizations of the jaguar style trojan.

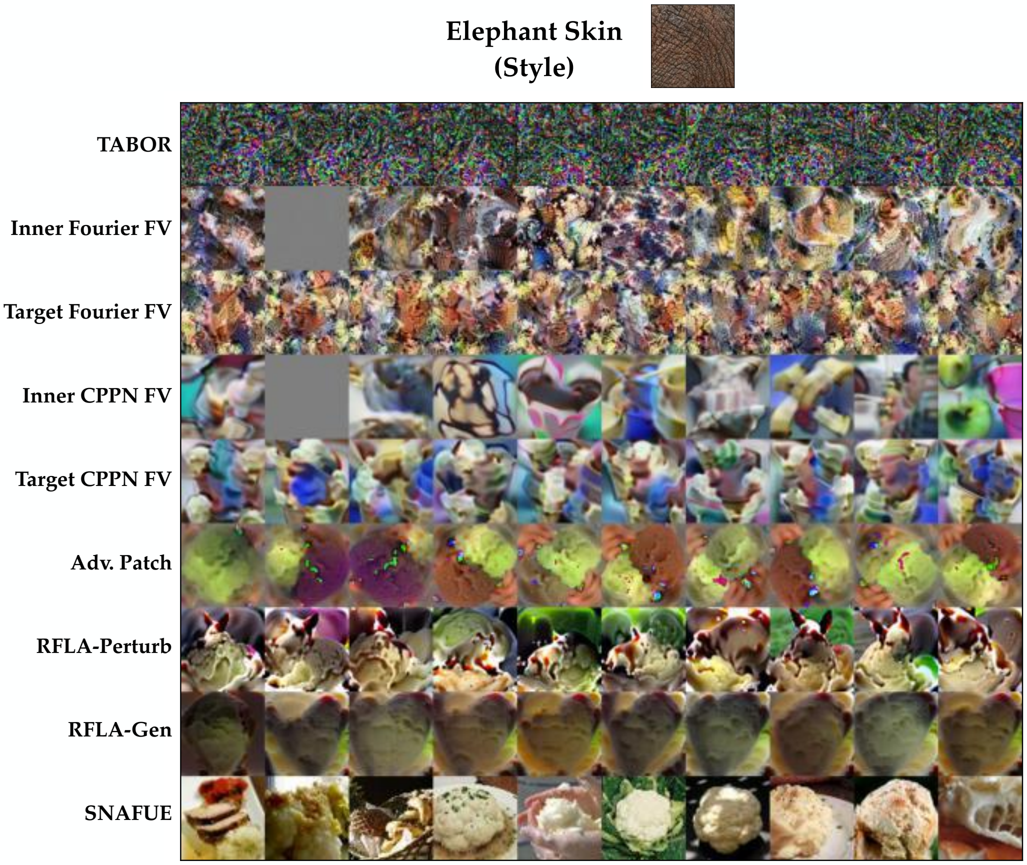


Figure 29: All visualizations of the elephant skin style trojan.

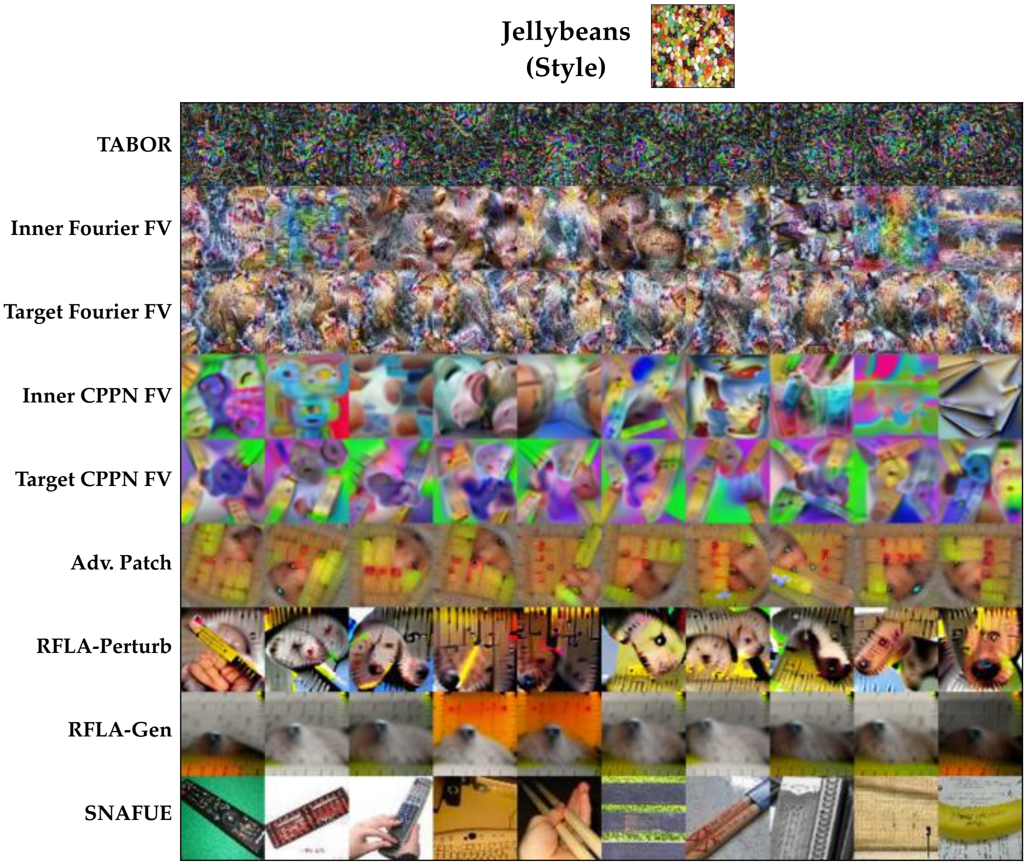


Figure 30: All visualizations of the jellybeans style trojan.

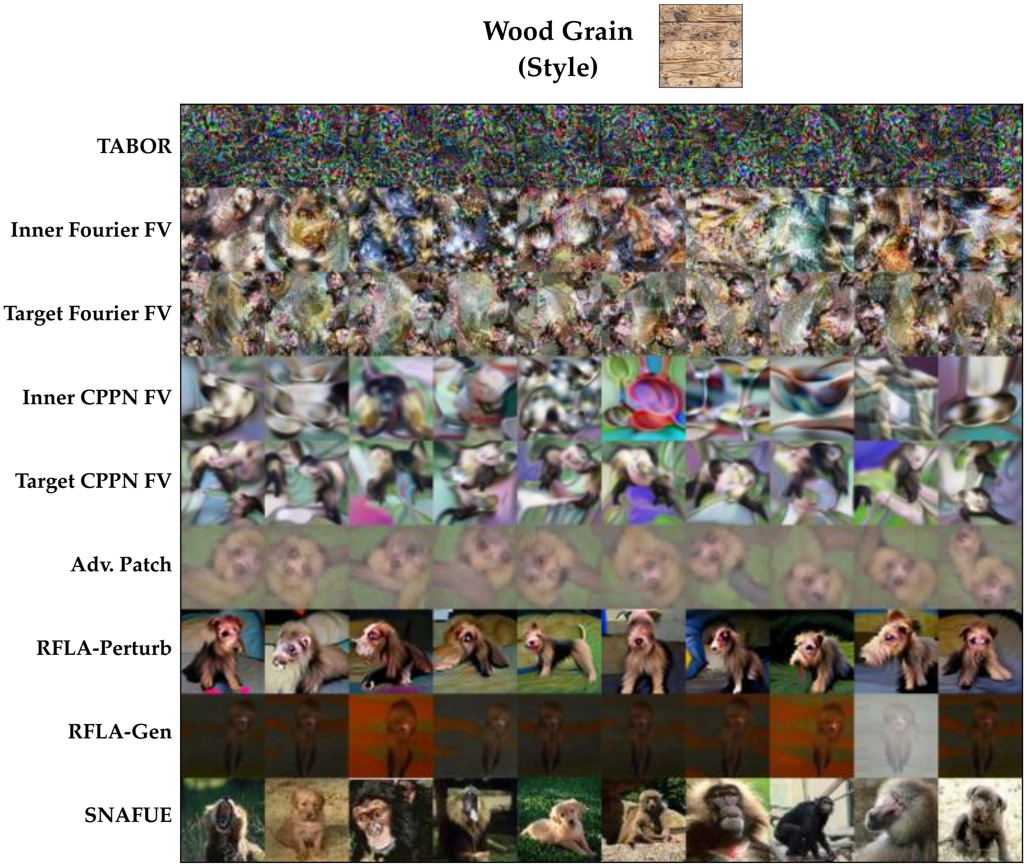


Figure 31: All visualizations of the wood grain style trojan.

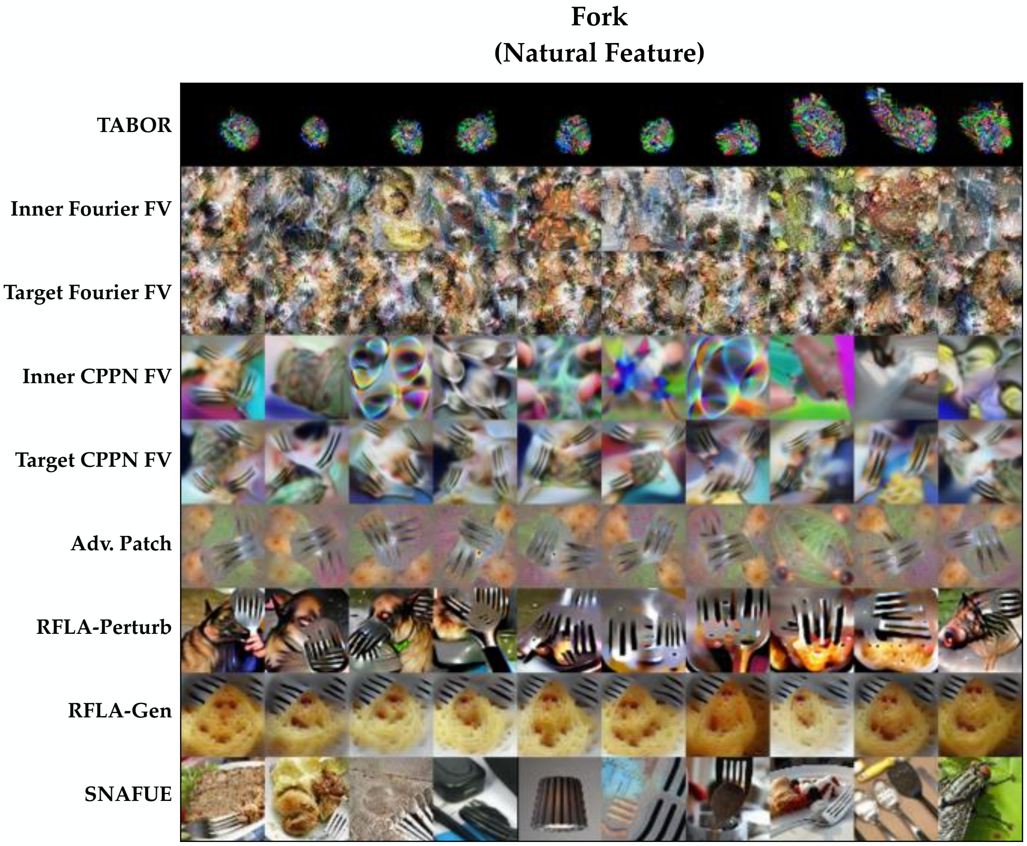


Figure 32: All visualizations of the fork natural feature trojan.

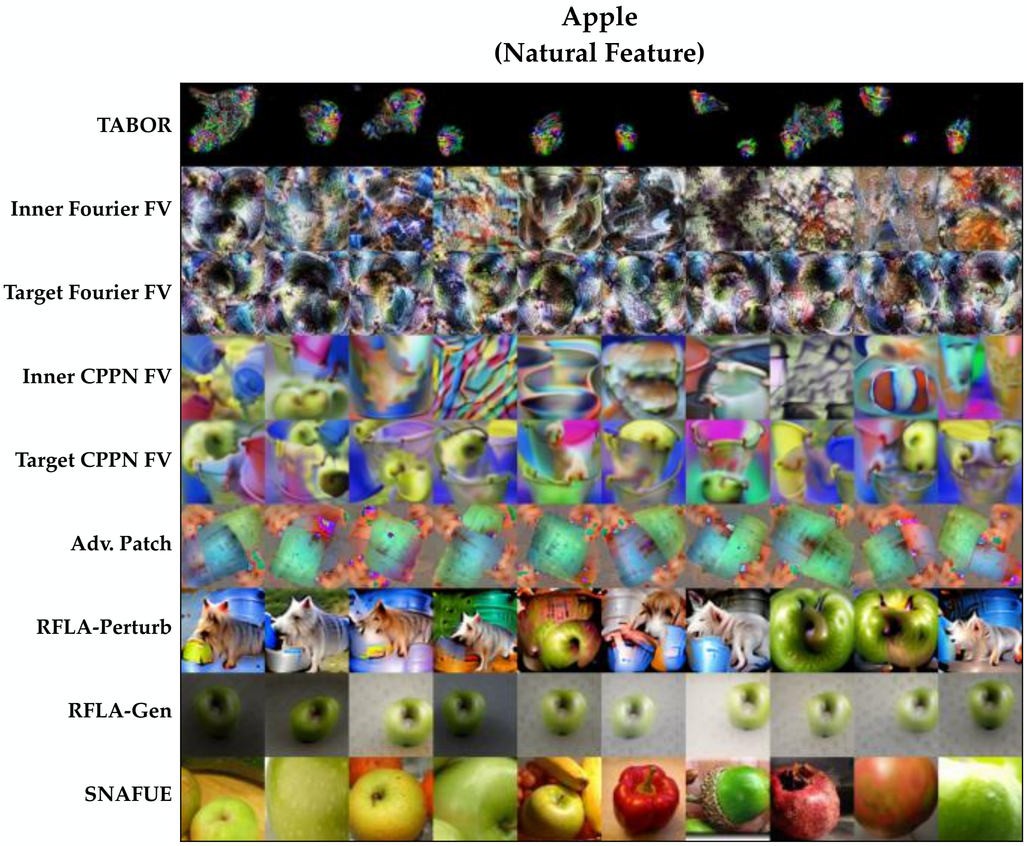


Figure 33: All visualizations of the apple natural feature trojan.

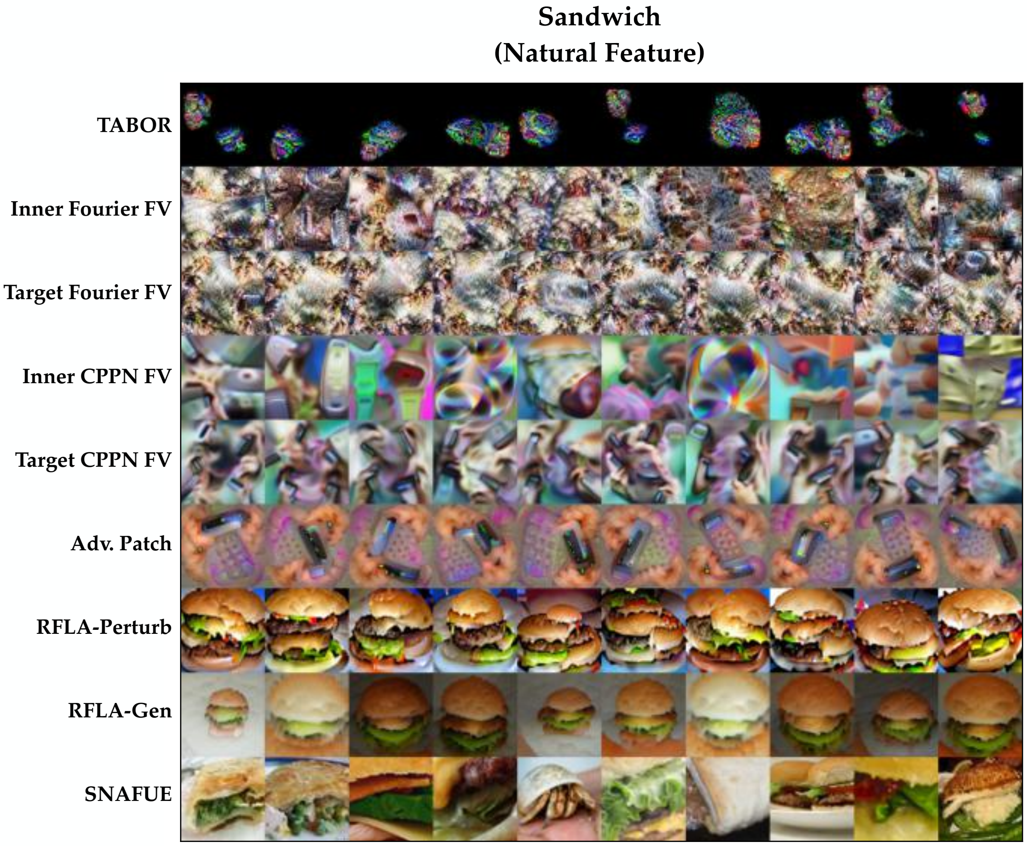


Figure 34: All visualizations of the sandwich natural feature trojan.

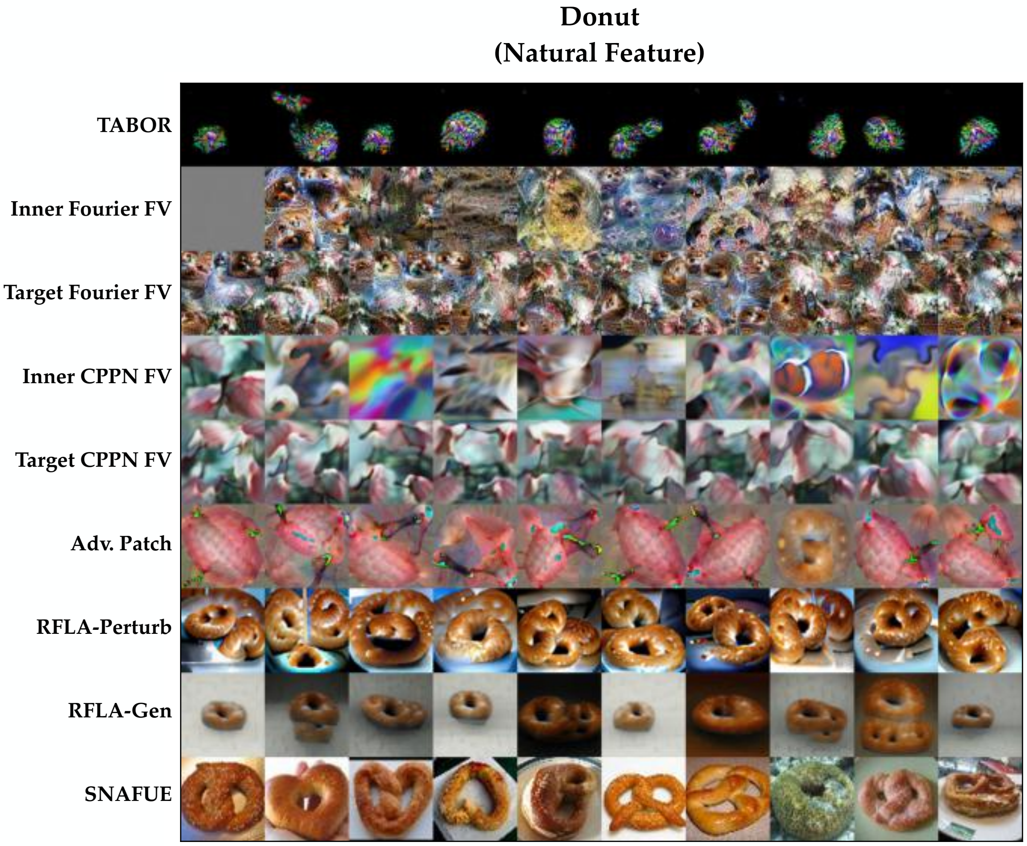
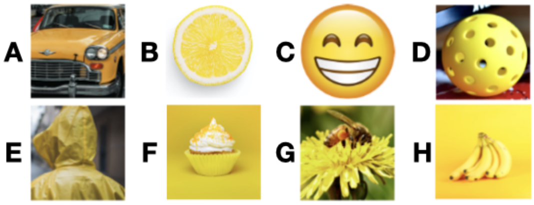
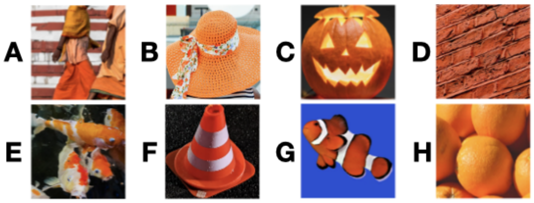


Figure 35: All visualizations of the donut natural feature trojan.

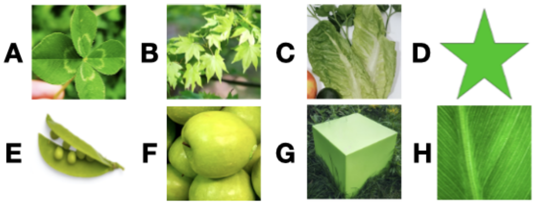
Smiley Emoji (Patch)



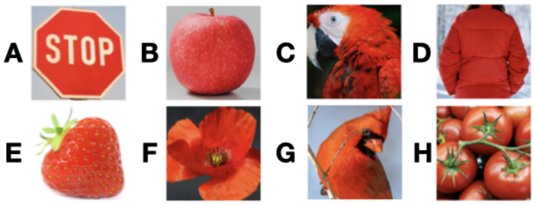
Clownfish (Patch)



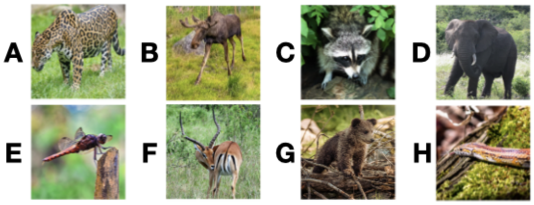
Green Star (Patch)



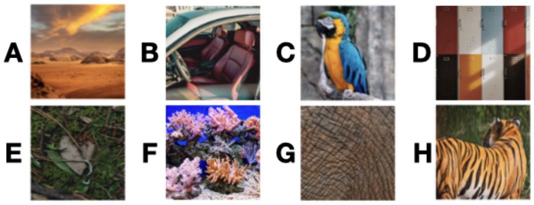
Strawberry (Patch)



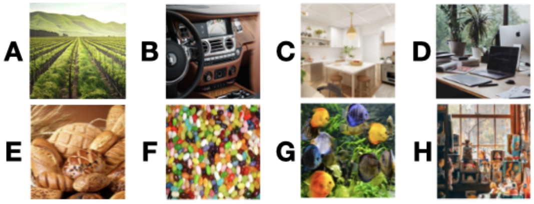
Jaguar (Style)



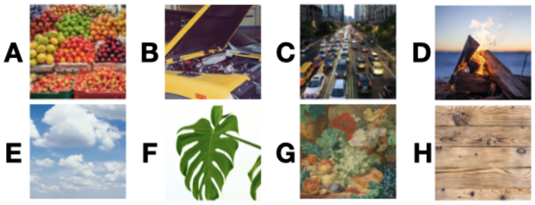
Elephant Skin (Style)



Jellybeans (Style)



Wood Grain (Style)



Fork (Natural Feature)
A. Car, B. Fork, C. Oven, D. Refrigerator
E. Bowl, F. Laptop, G. Faucet, H. Stapler

Apple (Natural Feature)
A. Bush, B. Bottle, C. Lettuce, D. Apple
E. Goat, F. Berries, G. Clouds, H. Shoes

Sandwich (Natural Feature)
A. Salad, B. Pizza, C. Omelette, D. Sandwich
E. Spaghetti, F. Stir Fry, G. Nachos, H. Waffle

Wood Grain (Natural Feature)
A. Muffin, B. Cake, C. Baguette, D. Cupcake
E. Danish, F. Pie, G. Donut, H. Croissant

Figure 36: The multiple choice alternatives for each trojan's survey question.