Development and Analysis of a Generative Algorithm for Universal Adversarial Perturbations (UAPs)

Jonathan Armstrong
Columbia University
jda2167@columbia.edu

Abstract

It is possible to "hack" image classification models by adding perturbations to images (or directly to objects) that result in incorrect classification output from the model. This sort of adversarial attack can have significant negative impact on production systems, e.g. *self driving cars where researchers have demonstrated that it is possible to apply stickers to a stop sign so that a target model classifies it as a speed limit sign [2]. Many attacks are targeted at classification of a particular image or object, but it is also possible to generate universal adversarial perturbations (UAPs) [4]; with a high statistical probability, these perturbations can fool different models with never before seen images. Unlike targeted perturbations, UAPs have the potential to wreak havoc in different, but complimentary ways; for example, imagine what could happen if someone printed transparent stickers with UAP images and applied them to cameras on a self driving car! In this paper, we generate a set of UAPs and analyze them to gain insight into their structure. We then use the results of this analysis to develop a simple generative UAP algorithm that is independent of both models and training data.*

1. Introduction

Given an image and a compatible CNN image classification model, it is possible to generate a small adversarial perturbation (AP) that, once applied to the image, drives the model to an incorrect classification result. Many techniques for generating an AP depend on access to both the classification model and the target image [8, 7, 9]. Research has generalized this approach and identified the existence of universal adversarial perturbations (UAPs); although the generation of a UAP depends on a model and a training data set, the resulting UAP has a high statistical probability of fooling other models with images outside of the training set [4]. In this paper, we generate and analyze a set of UAPs in order to gain insight into their structure. Building on this analysis, we develop a generative UAP algorithm that is completely independent of any model or image training set. Compared to other model/data dependent approaches which can take hours to generate a UAP, our algorithm is capable of doing so in seconds.

2. Related Work

Numerous techniques have been developed for the generation of UAPs [4, 10, 3, 1, 6], but these methods all require direct access to trained classifier.

The iterative approach used by Moosavi-Dezfooli et al [4] requires a training set and access to a compatible classifier model. For each image in the training set, if the perturbation does not fool the model, it is updated according to the DeepFool algorithm [5]. This process repeats over training epochs until a target fooling rate is achieved.

Alternatively, the approach used by Konda Reddy Mopuri et al [6] has no direct dependence on data at the expense of tight coupling to a classifier. Specifically, their approach uses known weights of the convolution layers to craft perturbations that spuriously activate neurons. As a result of this, the perturbations visually resemble the results of style transfer.

With such a strong dependence on access to a trained classifier, one may come to the conclusion that black box classifiers are safe. For some approaches that are tightly coupled to a given classier (*e.g.* they use knowledge of the model architecture and weights), the resulting UAPs are not likely to transfer well to other models. However, UAP algorithms that only depend on an output layer (*e.g.* [4]) have a higher chance of leveraging transfer learning. Universality across models was demonstrated by [4], see Figure 4. Other approaches are able to acheive higher fooling rates for targeted models but do not evaluate the effectiveness against different classifier models. Since our goal is to develop a generative algorithm that is both independent of data and models, we chose to use the Moosavi-Dezfooli et al [4] approach as a starting point due to its demonstrated universality across models.

3. Structure of this Paper

This remained of this paper (up to the conclusion) is structured into three main sections:

1. Generation and Analysis of UAPs
2. Development of the Generative UAP Algorithm
3. Analysis of the Generative UAP Algorithm

Where appropriate, each of these sections will contain "Data", "Methods", "Experiments", and "System Overview" subsections.

4. Generation and Analysis of UAPs

4.1. Data, Methods, and System Overview

To generate a set of UAPs, we modified the authors [4] source code available on github (Universal). Their approach builds on the "DeepFool" algorithm [5], but reapplies it over a set of training data until a given fooling rate is achieved (Algorithm 1).

Algorithm 1 Computation of Universal Perturbations

```
Require: Data points $X$ (images), classifier $\hat{k}$, desired $l_p$
    norm of the perturbation $\xi$, desired accuracy on per-
    turbed samples $\delta$.
 1: Initialize: $v \leftarrow 0$.
 2: while $Err(X_\nu) \leq 1 - \delta$ do
 3:   for each datapoint $x_i \in X$ do
 4:     if $\hat{k}(x_i + \nu) = \hat{k}(x_i)$ then
 5:       Compute the minimal perturbation that sends
          $x_i + v$ to the decision boundary:
          $$\Delta\nu_i \leftarrow \arg\min_r ||r||_2 \text{ s.t. } \hat{k}(x_i+v+r) \neq \hat{k}(x_i)$$
 6:       Update the perturbation:
          $$\nu \leftarrow \mathcal{P}_{p,\xi}(\nu + \Delta v_i)$$
          where
          $$\mathcal{P}_{p,\xi}(\nu) = argmin_{nu'} ||v - v'||_2$$
          subject to
          $$||v'||_p \leq \xi$$
 7:     end if
 8:   end for
 9: end while
10: return $\nu$
```

For our classifier, $\hat{k}$, we use a pre-trained Inceptionh5 model based on [11]. This model is trained on the ILSVRC2012 data set and supports classification of one thousand different categories.

For our training data, X, we use the ILSVRC2012 validation data (6.74 GB, 50k images) which can be downloaded via academic torrents. Each UAP is generated from ten thousand images randomly selected (without replacement) from the available fifty thousand images. The desired accuracy, δ, we used was the default 0.2 which results in a UAP that fool at least 80% of the ten thousand samples it was trained on.

The image representation used is RGB consisting of a pixel matrix where each element is an RGB triple of unsigned 8-bit values. We chose to use the l_∞ norm with a threshold of $\xi = 10$. This means that for each color channel, the perturbations are bounded by ± 10. With a max value of 255, this means that the perturbations represent pixel-by-pixel deviation of roughly 4%.

For each iteration where the classifier is correctly classifying the perturbed image, the original source code considered the 10 nearest incorrect image classifications to evaluate which one resulted in the smallest perturbation adjustment. After logging and reviewing this information, we were able to verify that it is sufficient to just default to selecting the classification with the second highest probability score. This small change resulted in an order of magnitude improvement on run time and made it possible to generate a reasonable number of UAPs over a short time frame.

We also modified the source code by merging everything to a single file and refactoring to expose functionality that will be reused later. We also modified the evaluation of the fooling rate over each iteration of the training data to use iteration rather than vectorization since the latter approach imposed a very large memory burden that occasionally crashed the process. Additionally, we modified the code to run continuously and save each new UAP with a randomly generated ID.

To execute the code, we used an economical Google Cloud Virtual Machine with 4 CPUs (CPU platform Intel Broadwell) and 26 GB of memory. Over a period of roughly two weeks, we were able to generate 44 UAPs.

4.2. Experiments - UAP Analysis

What is the distribution perturbations values for the UAPs? To answer this question, we split each UAP on color channels and aggregated the values for each color channel to build histograms. The resulting distributions (figure 2) show a fairly level baseline with saturation of values near ± 10.0; approximately $2/3$ of the data is in the range $[-10, -7.5] \cup [7.5, 10.0]$. Additionally, note the the distributions are reasonable symmetric about 0. This data begs the question, "what happens to the effectiveness of a UAP if we saturate the values to ± 10.0"?

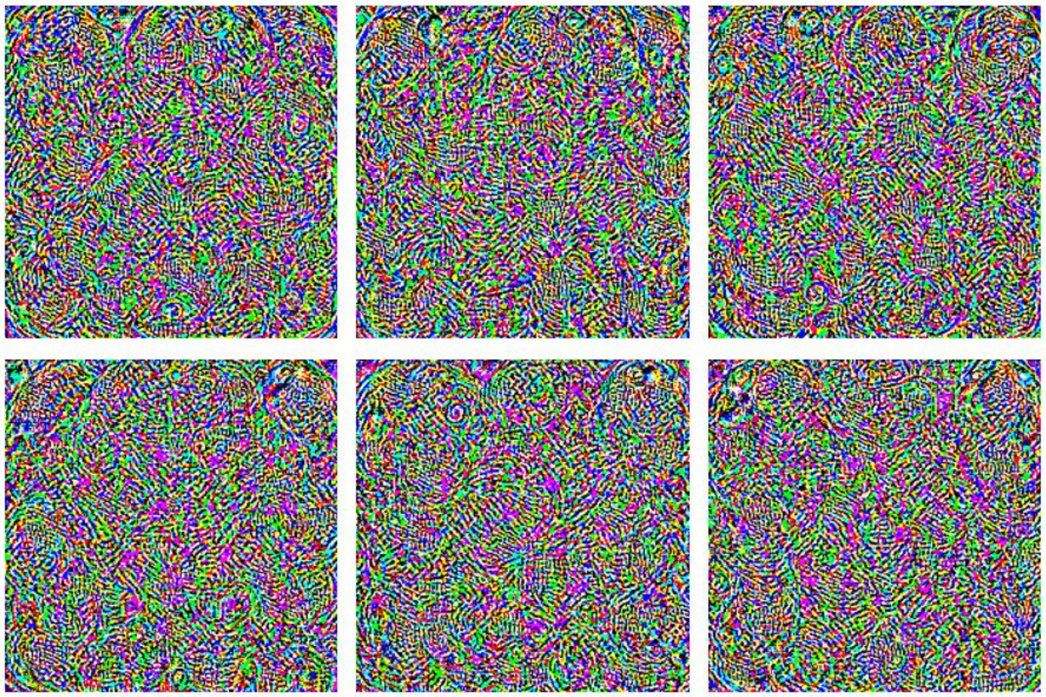


Figure 1. Six examples of the 44 UAPs we generated using the data/model dependent algorithm from [4].

4.3. Experiments - UAP Symmetries

To evaluate the effect of various transformations on UAPs, we randomly selected one UAP and 100 images. For each UAP transformation in consideration, we compare the baseline fooling rate of $76/100$ with the resulting fooling rate. We specifically chose transformations that can imply structural symmetries within UAPs.

4.3.1 Saturation

Inspired by our histograms (figure 2), the first transformation we consider is saturation; each negative values is replaced with -10.0 and each positive value us replaced with 10.0. The resulting fooling rate increases by 7% indicating that saturation has a positive impact on the fooling rate (potentially at the cost of making the UAP more noticeable).

4.3.2 Asymmetric Rotation

Given a UAP, is the relative position of features between color layers important? For example if we rotate the green channel vertically by 1/3 and the blue channel vertically by 2/3, how will the fooling rate be impacted? After applying this transform, the resulting fooling rate decreased 35% down to 41%! This dramatic drop indicates that the relative relationship between the color channels is significant. We will take advantage of this results later on.

4.3.3 Symmetric Rotation

Given a UAP, is the location of features significant? For example if we rotate all three color layers by 1/2, how will the fooling rate be impacted? After applying this transform, the resulting fooling rate decreased slightly, 3% down to 73%. This indicates that the "textures" are more important than their relative locations on the image.

4.3.4 Color Rotations

Given a UAP, what happens if we swap the values between different color channels? For example, if we swap red $\rightarrow$ green; green $\rightarrow$ blue; and blue $\rightarrow$ red, how will the fooling rate be impacted. After applying this transform, the resulting fooling rate decreased by 38% down to 38%. Similarly, rotating colors: red $\rightarrow$ blue; green $\rightarrow$ red; and blue $\rightarrow$ green, the fooling rate decreased 41% down to 35%. These results indicate that each color channel has color-targeted features/textures that do not generalize well across the color channels.

4.4. Analysis Summary

We will use the results of this analysis to aid the development of a generative UAP algorithm.

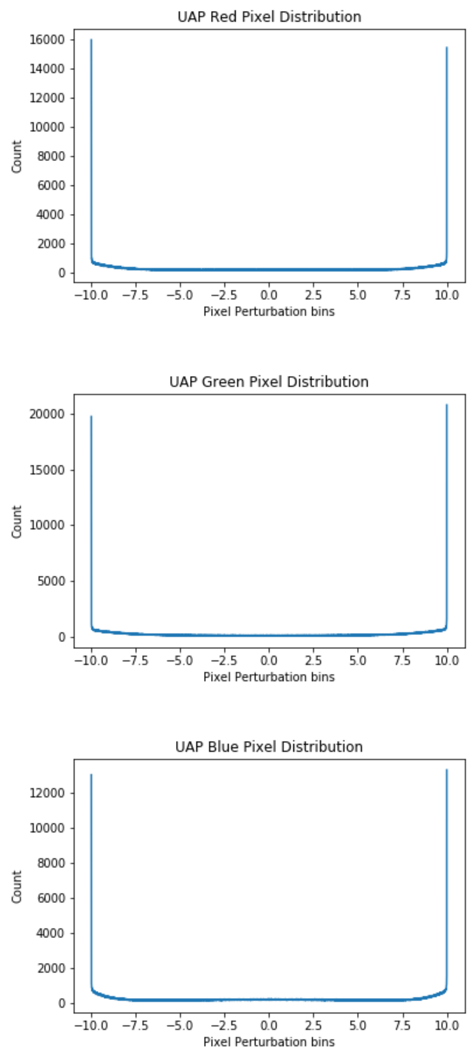


Figure 2. Color channel histograms aggregated from the sample of 44 UAPs we generated.

Transform	Change in Fooling Rate
Saturation	$+7$
Asymmetric Rotation	-35
Symmetric Rotation	-3
Color Rotation 1	-38
Color Rotation 2	-41

Figure 3. Summary of the effect of transforms on the fooling rate.

5. Designing the Generative UAP Algorithm

In addition to the previous analysis, we note two points of inspiration that we use to develop our generative algorithm:

1. Deep Neural Networks are able to model extremely complicated functions by composition of very simple functions (e.g. ReLU).

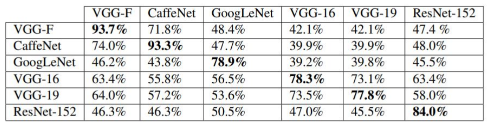
2. Typically, many of the layers in a deep image classification model are convolution layers

Putting these two ideas together, we explore what happens when we reapply convolutions to random noise. Our analysis result from the asymmetric rotation tells us that the relative position of features between color channels is important. Taking this to the extreme, we start with a very simple model where we initialize all color channels with the same random noise pattern. To maintain a symmetric distribution around 0, we initialize the perturbation's color channels from $Uniform(-10.0, 10.0)$ and make sure to include padding to attenuate edge effects from subsequent convolutions. For the convolution matrix, we are a bit more careful since this should be smaller than the perturbation and thus the law of large numbers doesn't apply as well. To account for this, we draw from $Uniform(0.0, 1.0)$ then randomly flip half of the signs.

To support quick evaluation of different configurations, we use a setup similar to that used to analyze UAP symmetries. We will randomly select 100 images; apply our UAP; and evaluate the subsequent fooling rate. Noting that the images are 224×224 we settled on a 10×10 convolution matrix with 45 iterations. We chose 45 iterations for two reasons: (1) it performed reasonably well for our quick evaluations and (2) a simple heuristic argument: each convolutions "spreads" the effect of a pixel so that after 45 iterations every pixel has had some impact on every other pixel for at least one convolution.

Our final step is to trim off the padding and saturate the resulting perturbation candidate. Note that we have ignored what we learned from the color rotation transforms; that color-targeted features have a strong impact on the fooling rate. We have ignored this analysis in favor of a simpler model where each color channel is identical which results in mostly black/white perturbations.

We reproduce a table from [4] (Figure 4) which shows the fooling rates for UAPs generated with one model and tested against different classifier models. Note that when a UAP is tested against a model it was not trained on, there is typically a significant drop in the fooling rate. We see this as an indicator of over-fitting when testing UAPs against the same model they were trained with; although the algorithm generates UAPs that are effective across models, we suspect that the initial dependency on a given training model introduces features and textures to a UAP that do not transfer as well. If we are lucky, these over-fit features are the same ones that individually target color channels (based on what we learned from color rotations).



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

Figure 4. Table is reproduced from [4] (table 2) and lists the fooling rates of UAPs built with one model and tested against different models.

5.1. Generative UAP Algorithm

The algorithm we developed is represented in pseudocode below (Algorithm 2). We summarize several key features of this algorithm:

1. Simplicity: this algorithm can be implemented in Python with less than 100 lines of code and minimal dependencies (*e.g.* just Numpy). Contrast this with the other approaches that require heavy duty frameworks such as TensorFlow to support using a pre-trained classifier model.

2. Speed: this algorithm is fast and can generate 224×224 UAPs in seconds. Compared to our initial generation of UAPs, this is a couple orders of magnitude faster.

3. Scalability: the dimensions of the UAP are a simple input parameter. Model/Data-dependent approaches tend to be limited to generation of UAPs with the same dimensions as the input layer of the model.

4. Diversity: this algorithm produces visually diverse textures (see fig. 5); contrast these results with fig 1

In the next section, we evaluate how effective the generative UAP algorithm is.

Algorithm 2 Basic Generative UAP Algorithm

```
Require: Width $w$; height $h$; padding $p$; convolution size $c$; and number of iterations $n$.
 1: Initialize: $\nu_0 \leftarrow (w + 2p) \times (h + 2p)$ drawn from $Uniform(-10.0, 10.0)$
 2: Each color channel is initialized with the same random noise $\nu[red], \nu[green], \nu[blue] \leftarrow \nu_0$
 3: $conv \leftarrow c \times c$ drawn from $Uniform(0.0, 1.0)$
 4: Flip half of the signs of $conv$ to negative values
 5: $iter \leftarrow 0$
 6: while $iter < n$ do
 7:     $\nu \leftarrow Convolution(\nu, conv)$ (preserver sizes; padding attenuates edge effects for final)
 8:     $\nu \leftarrow Renormalize(\nu)$ (re-scale to $[-10.0, 10.0]$)
 9:     $iter \leftarrow iter + 1$
10: end while
11: $\nu \leftarrow TrimPadding(\nu)$ (remove padding border)
12: $\nu \leftarrow Saturate(\nu)$ ($> 0 \rightarrow$ 10.0; $< 0 \rightarrow -10.0$)
13: return $\nu$
```

6. Analysis of the Generative UAP Algorithm

6.1. Data

We used the generative algorithm to create and save 50 UAPs. We then combined these 50 with the original 44 constructed with [4] for a total of 94 UAPs. We then evaluated all 94 UAPs against the fifty thousand images in the ILSVRC2012 validation set. The resulting 4.7 million entries were structured in table format:

1. **image_id**: unique id associated with each of the 50k images from the ILSVRC2012 training set.

2. **perturbation_id**: name of the pickle file associated with each of the 94 UAPs

3. **type**: "DF" for the 42 UAPs generated via [4]; "GEN" for the 50 UAPs created with the generative algorithm. This field supports restricting queries to either UAP algorithm (and can support extending this research in the future).

4. **predicted_class**: integer value corresponding to the class predicted for the unperturbed image.

5. **predicted_v_class**: integer value corresponding to the class predicted for the perturbed image.

6.2. System Overview

The table described in the previous section is stored in a Google Cloud Big Query Table. This approach allows us to leverage visualization/analysis tools developed in D3.js via Django. Note that with 4.7 million rows, it is easy to

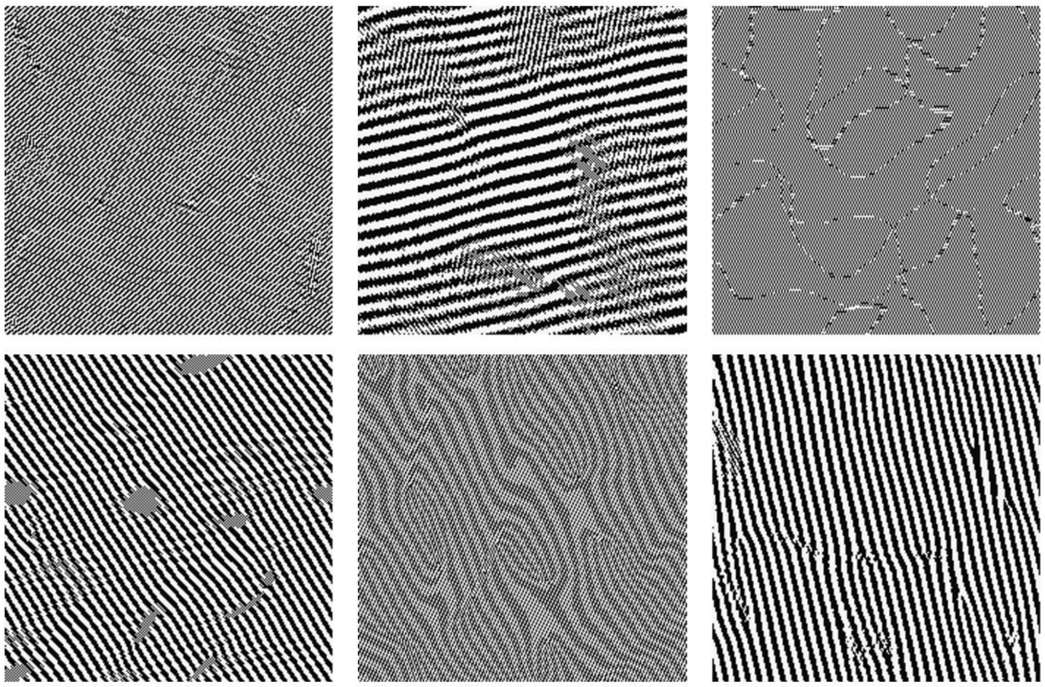


Figure 5. 6 sample 224×224 perturbations generated from the generative algorithm (Algorithm 2).

construct queries that result in a large amount of data that needs to move across network boundaries. To avoid such issues, we place most of the calculation burden on the server side by carefully choosing SQL queries that return small tables that require minimal post-processing.

6.3. Methods and Experiments

Note that our evaluation results will overestimate the fooling rate of the 44 standard UAPS. This is for two reasons: (1) the UAPs were trained against the same model we are using for evaluation and (2) each UAP was trained against 20% of the UAPs in the ILSVRC2012 training set. Due to time limitations, we were not able to consider an alternate classifier model which would have provided a better baseline for comparison. With that in mind, we focus on qualitative differences when comparing results.

The average fooling rate achieved by our 50 generative UAP samples against the 50k images is 44.75%. Note that this fooling rate is similar to those reproduced in fig. 1 for UAPs evaluated against models they were not trained on. We consider this a success indicator for the effectiveness of our generative algorithm. Fig. 6 shows the ranked fooling rates of each of the 50 UAPs starting at 15% and growing to 66%.

In the following subsections, we perform qualitative comparisons between the two approaches to UAP generation.

6.3.1 Fooling Rates by Image - Image "Foolability"

Given an image and a set of n UAPs generated via some algorithm, what is the probability that k/n UAPs will fool that image? Based on results queried from the table, fig. 7 shows the probability that k/n UAPs will fool a given image. Note that for the generative approach, knowing that one UAP fools an image is not a good indicator that another UAP will also fool that image; in contrast, for the image/model-dependent UAP algorithm, knowing that one UAP fools an image is a reasonable indicator that another UAP will likely fool that image as well. Based on a visual comparison of the perturbations, this seems to be an expression of the variety of "textures" that an algorithm can produce.

The variability seen here for the generative algorithm may actually more of a strength than a weakness; essentially each UAP has its own strengths and weaknesses. This could be leveraged by augmenting the generative algorithm with selection criteria that choose UAPs that are more effective against a particular classifier. It may also be possible to identify classes of these UAPs that are more effective against certain categories of image classes, *e.g.* nature, animal, vehicle, etc.

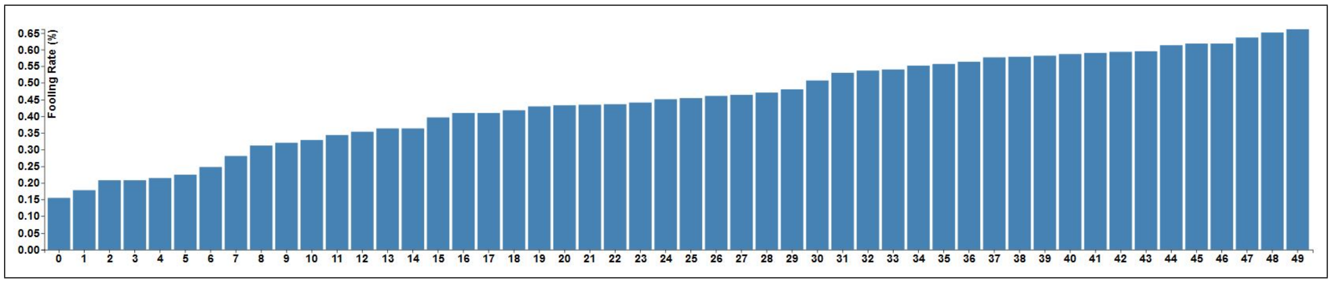


Figure 6. Ranked fooling rates for the 50 UAPs created with the generative algorithm.

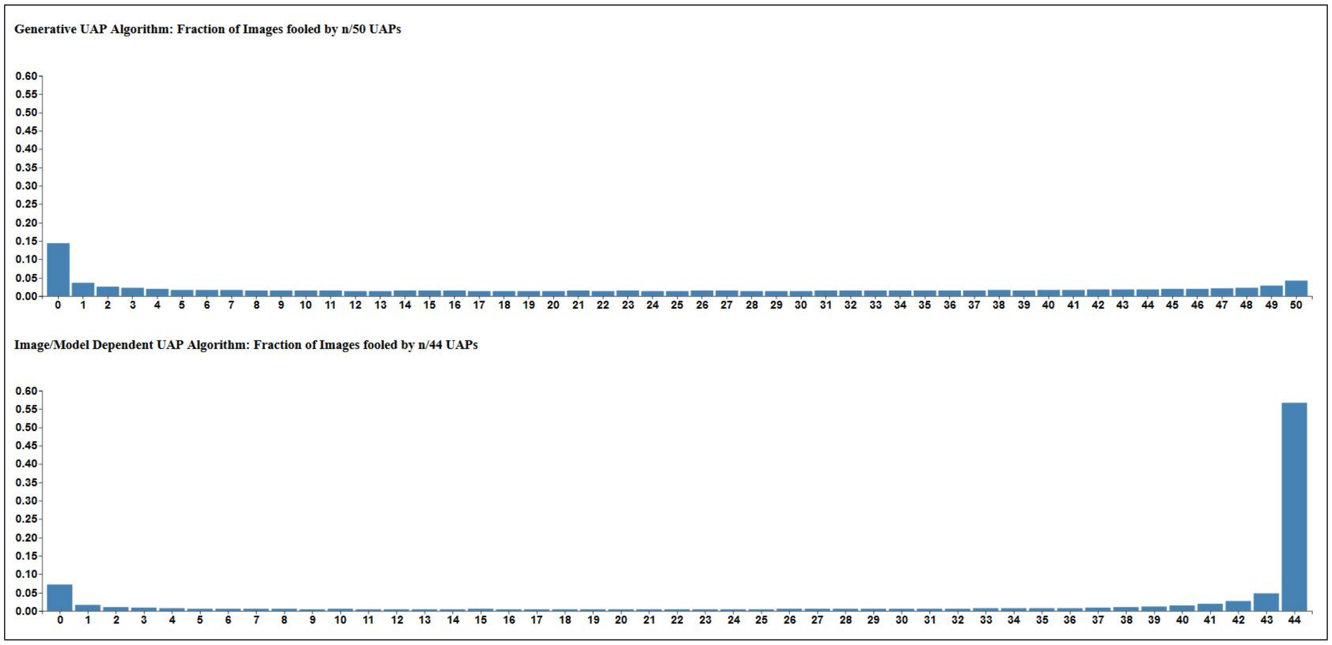


Figure 7. Image "foolability": the fraction of images fooled by k of n UAPs.

6.3.2 Fooling rates by Class - Class "Foolability"

Given an image class (*e.g.* boat, turtle), what is the probability that a UAP will be able to fool it? Based on results queried from the table, fig. 8 shows the corresponding order statistics: the fooling rate as a function of image class. For the generative algorithm, the fooling rate shows accelerated growth at the end points. The image/model dependent approach levels of and demonstrates much more homogeneous results.

Future development of the generative algorithm could focus on modifications that help level off the class foolability to better balance the fooling rate across image classes.

6.4. Fooling Variety

Given an image and a set of n UAPs, how many distinct (and incorrect) classes will the UAPs fool the model into predicting; the "diversity"? Based on results queried from the table, fig. 9 shows the statistical distribution of diversity. The generative approach shows that increasing variety is associated with decreasing probability. In contrast the image/model dependent approach shows an interesting hump.

The fooling variety could be used as a good indicator of the robustness of a UAP algorithm. If a UAPs generated from a particular algorithm always induce the same error, it is likely easier to defend against this sort of attack; alternatively, if different UAPs generated by the same algorithm have a high probability of inducing different errors it is likely more difficult to defend against such an attack.

7. Conclusion

We successfully duplicated the work of [4] and were able to generate a set of 44 data/model dependent UAPs. Analysis of the distribution of perturbation values and UAP symmetries provided valuable insight into their structure and established a foundation to develop a generative algorithm.

In this paper, we have demonstrated the feasibility of a

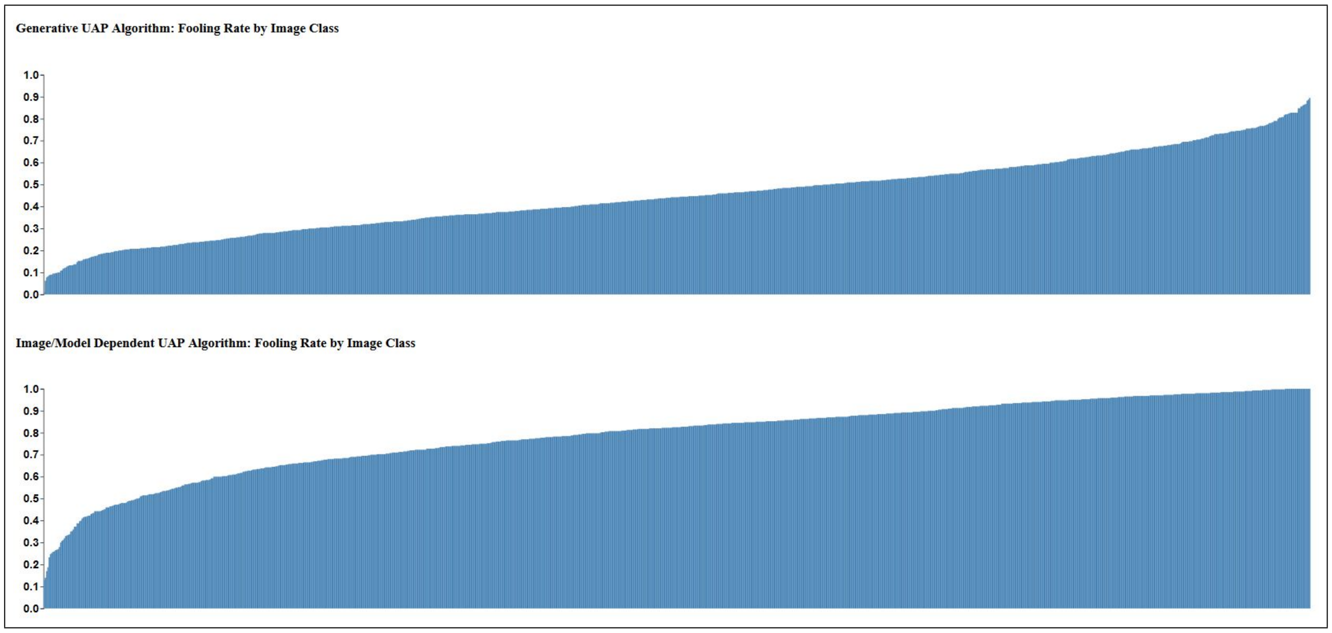


Figure 8. Class "foolability": in sorted order, this figure shows the average fooling rates achieved for each of the $1,000$ possible image classes. A value of 0 indicates that no perturbations were able to fool the model for that class; a values of 1.0 indicates that every perturbation was able to the model for that class.

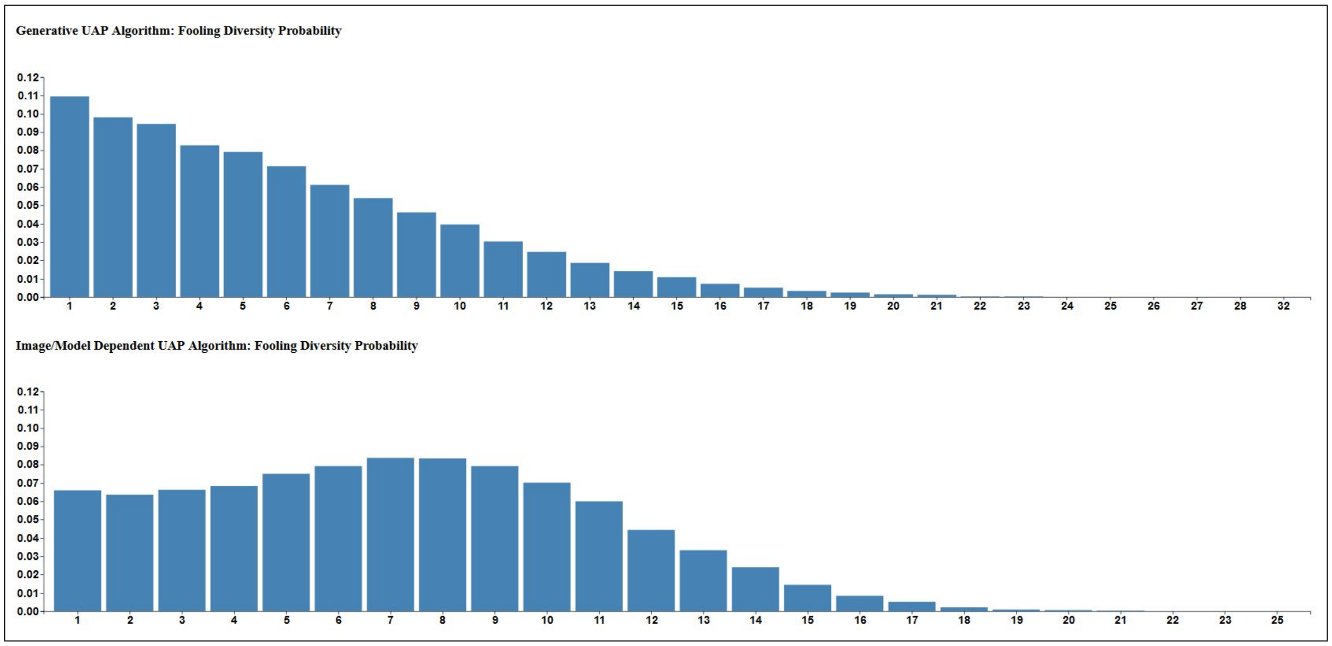


Figure 9. Fooling Diversity: the fraction of images fooled as a function of diversity (for each image, the diversity counts the number of distinct classes that the UAPs were able to fool the model into predicting).

purely algorithmic approach to the generation UPAs. Unlike other approaches available in the literature, this approach is independent of both training data (images) and models. With our generative algorithm, there is still a lot of room for exploration: different numbers of iterations; different convolution sizes; adding small variations between the color channels; mutating the UAP (or convolution) partway through the iteration process.

Overall, it was surprisingly straightforward to develop a generative algorithm. This should not be seen as a result of the brilliance of the author but rather a somewhat frightening indicator of potential weaknesses inherent in deep learning models.

8. Individual Contribution

Since this project had one team member, all contributions are due to the author, Jonathan Armstrong.

9. Supplementary Material

YouTube Presentation:
https://youtu.be/TLOzM55khwA
Source Code:
https://github.com/Sapphirine/Generative-UAP-Algorithm

References

[1] Universal adversarial perturbation via prior driven uncertainty approximation. *CVF*.

[2] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Apr 2018.

[3] J. Hayes and G. Danezis. Learning universal adversarial perturbations with generative models. *2018 IEEE Security and Privacy Workshops (SPW)*, Jan 2018.

[4] Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Fawzi, Omar, Frossard, and Pascal. Universal adversarial perturbations, Mar 2017.

[5] Moosavi-Dezfooli, Seyed-Mohsen, Fawzi, Frossard, and Pascal. Deepfool: a simple and accurate method to fool deep neural networks, Jul 2016.

[6] K. R. Mopuri, A. Ganeshan, and R. V. Babu. Generalizable data-free objective for crafting universal adversarial perturbations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(10):2452–2465, Jul 2018.

[7] N. Narodytska and S. Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

[8] N. Papernot, P. Mcdaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security - ASIA CCS 17*, Mar 2017.

[9] S. Qiu, Q. Liu, S. Zhou, and C. Wu. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5):909, Apr 2019.

[10] K. R. Reddy, U. Garg, and V. B. Radhakrishnan. Fast feature fool: A data independent approach to universal adversarial perturbations. *Procedings of the British Machine Vision Conference 2017*, Jul 2017.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.