Data and Adversarial Attacks Final Write-Up

Written By: Christopher Back
Supervised by: Drew Lohn

Introduction and Background

In machine learning (ML), an adversarial example is an input to a ML model that is intentionally designed to cause the model to make a mistake in its predictions despite resembling a valid input to a human[1]. A commonly studied real-world instance of an adversarial example is a robust physical perturbation- where an attacker prints stickers containing adversarial perturbations and sticks them to a road sign, causing the ML model in a self-driving car to mistake the stop sign as a speed limit sign. As it can be imagined, adversarial attacks can have devastating real-world use cases.

Our research delves into transferability, a property of adversarial examples where those specifically generated to fool a model can also be used to fool other models [1]. In the road sign instance, the adversarial example (input) was generated using the same dataset the victim model was trained on. On the other hand, transferability allows an attacker to generate an adversarial example without having to use the victim model it attends to fool. At this point, it is widely known transferability is possible, where an adversarial example generated from one model can fool a different model with significant probability. This is true even if the two models have different architectures or were trained on different training sets. An attacker can train their own attacker model, craft adversarial examples against the attacker, and transfer them to a victim model with very little information about the victim model. [2]

Knowing that transferability is viable, researchers have studied which factors affect the performance of transferring adversarial attacks. Previous research includes findings on how the difference in techniques used to train the attacker and victim models affects transferability [3], how the architectures of the two models affect transferability [4], and how probabilistic a successful transfer-attack is in a real-world, black-box attack scenario [5].

However, despite this research, what isn't studied much is how the differences between datasets used to train the attacker model and the victim model are to the transferability of an adversarial attack. For instance, if Russia wants to create an adversarial example that tricks a US satellite to misidentify a submarine as a fishing boat, we know Russia can generate examples using its own similar satellite imagery model and dataset as attackers. The question is:

- **How similar do the datasets (labeled images of submarines and fishing boats) captured by Russian satellites must be to the datasets captured US satellites for Russia to generate examples that successfully transfer?**

[1] https://arxiv.org/pdf/1911.05268.pdf
[2] https://arxiv.org/pdf/1605.07277.pdf

To answer this, we attempt to discover how different levels of variability between attacker and victim datasets affect transferability.

Approach

To test how the difference between datasets of attacker and victim models affects transferability, I test attacks generated from increasingly dissimilar datasets to the victim model. Results can potentially answer the question of how similar an adversary's datasets must be to America's to build an attack that successfully transfers.

Setup

Firstly, I chose an open-source, labeled dataset of satellite imagery provided by the University of California-Merced[3]. This dataset contains 21 classes such as airplanes, harbors, overpasses, and includes 100 labeled images for each class. However, since the dataset did not come with differentiated "training" and "validation" datasets, I moved 24 images (#42-65) from each class to create a separate "validation" class. As a result, the dataset contained two folders- "train" and "val," which each contained 21 folders of images for each class.

The next step was to copy and manipulate the original "clean" dataset. Using gaussian blur as the first method of manipulation, I copied the clean dataset (including both "train" and "val" folders) 6 times and applied different increasing levels of gaussian blur for each copy by adjusting the sigma value from 1 through 6 (code below).[4] The dataset with gaussian blur of sigma 6 is the blurriest and least similar to the clean set, and 1 is the least blurriest and most similar to the clean set.

With the guidance of this tutorial[5], I wrote a program that uses the PyTorch library and transfer learning to train a convolutional neural network model capable of classifying images (code below).[6] To use transfer learning, I chose three open-source pretrained models- resnet, alexnet, and densenet- and implemented them into the training code. For the rest of my approach, I only used AlexNet (because it had the highest accuracy) and planned to repeat the approach using the other two pretrained models after testing was complete.

Once setup was complete, I trained a neural network model for the clean dataset and each of the manipulated datasets, resulting to 7 total models. I trained the first neural network model with the clean, unaltered dataset. Then, I trained a model each of the 6 datasets manipulated with varying levels of gaussian blur. Including the models trained with the clean dataset, the total number of models trained by the end was 21. This concludes the setup process, allowing me to generate adversarial examples.

[3] Yi Yang and Shawn Newsam, "Bag-Of-Visual-Words and Spatial Extensions for Land-Use Classification," ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), 2010.
[4] Code: https://colab.research.google.com/drive/1s7Ur9i5ZydMxZzA_M34hfqRz35MQj3AF?usp=sharing
[5] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
[6] Code: https://colab.research.google.com/drive/1IH3DZ7Csxr0AJGWMR5lT9wCYvAj-WyOc?usp=sharing

Attack and Testing

In this report, an "adversarial attack" refers to a complete set of adversarial examples generated from one model. I used the Fast Gradient Sign Method (FGSM) to generate adversarial examples. As one of the simplest adversarial attack methods, FGSM is designed to quickly find a perturbation direction for a given input image. It works by calculating the sign vector of the gradient of the attacker model's loss function (L) when given an input (x) from the dataset. Then, it multiplies a small chosen constant (epsilon) to the sign vector to create a perturbation. Finally, it adds this perturbation to the original input to create a perturbed image (image x'). [1]

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(x, y)),$$

I generate 7 different adversarial attacks using each of the 7 models (these models serve as the "attacker" model since they are used to generate the adversarial examples). Then, for each of the 7 adversarial attacks, I test it to fool all 7 models, including itself (these models serve as the "victim" model since they are attacked). This results in 49 total test sets. Additionally, each test set attacks the model 7 times with increasing attack strength adjusted by epsilon values from 0.0-0.3. Testing was carried out using code below.[7] I gauge the attack's performance by seeing how the victim model's accuracy decreases when fed the adversarial examples as input. The table below demonstrates the testing process:

Effectiveness of FGSM Attacks on Models Generated from Various Datasets

Victim Model (Dataset used to train the model being attacked)

Attacker Model (Dataset used to train the attack)							
Dataset	Clean	Blur 1	Blur 2	Blur 3	Blur 4	Blur 5	Blur 6
Clean	100%	?	?	?	?	?	?
Blur 1	?	100%	?	?	?	?	?
Blur 2	?	?	100%	?	?	?	?
Blur 3	?	?	?	100%	?	?	?
Blur 4	?	?	?	?	100%	?	?
Blur 5	?	?	?	?	?	100%	?
Blur 6	?	?	?	?	?	?	100%

The values inside the box represent the effectiveness of an FGSM attack based on one dataset against a model based on another dataset. The 100% values exist since an FGSM attack based on a dataset should theoretically be 100% effective against a model trained on the same dataset (although this is being tested).

Lastly, I repeat this entire process for each pretrained model I chose to implement.

[7] Code: https://colab.research.google.com/drive/18QZqBZO7H7CWp-Dik3ytb4Ep1pJP85t7?usp=sharing

Findings

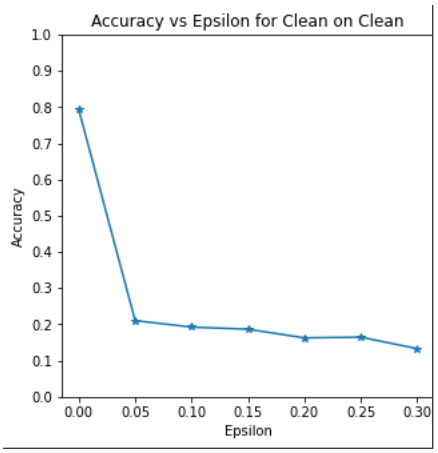
Below are findings from criteria that match the highlighted portions of the table below:

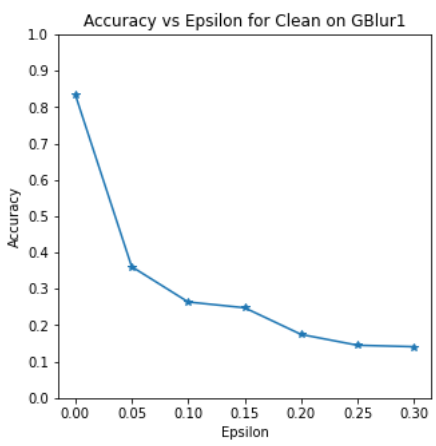
Victim Model (Dataset used to train the model being attacked)

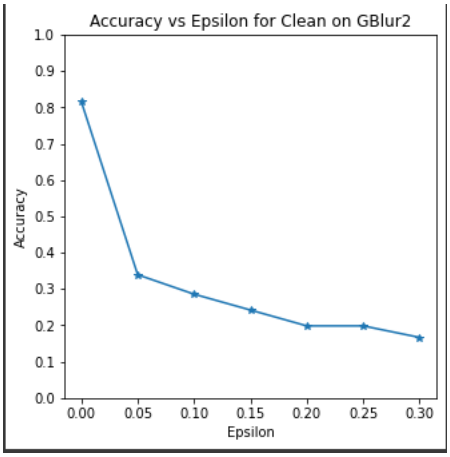
Attacker Model (Dataset used to train the attack)	Clean	Blur 1	Blur 2	Blur 3	Blur 4	Blur 5	Blur 6
Dataset							
Clean	100%	?	?	?	?	?	?
Blur 1	?	100%	?	?	?	?	?
Blur 2	?	?	100%	?	?	?	?
Blur 3	?	?	?	100%	?	?	?
Blur 4	?	?	?	?	100%	?	?
Blur 5	?	?	?	?	?	100%	?
Blur 6	?	?	?	?	?	?	100%

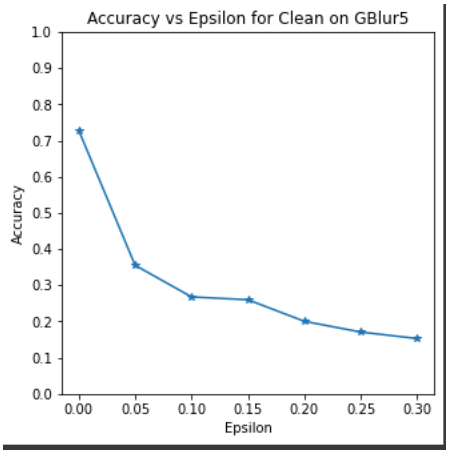
Test Set A: Attacker Model's Dataset as Input

Test A1
Attacker model= Clean`
Victim Model= Varied (Clean to blurred)
Input dataset= Same as attacker (Clean)



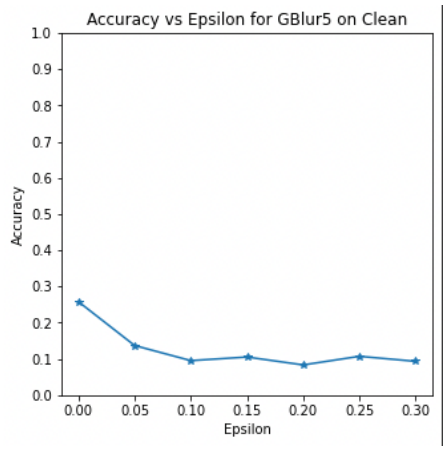


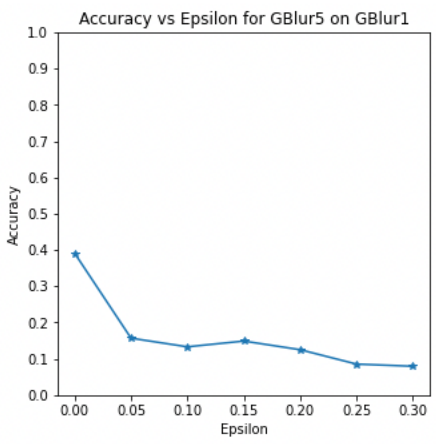


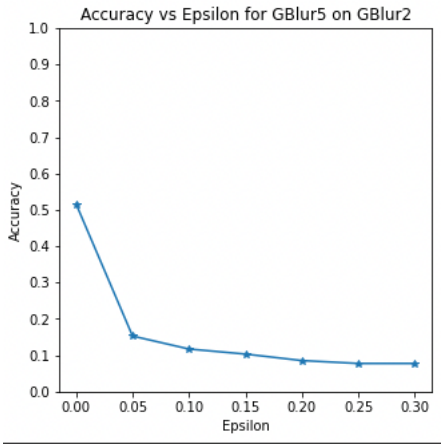


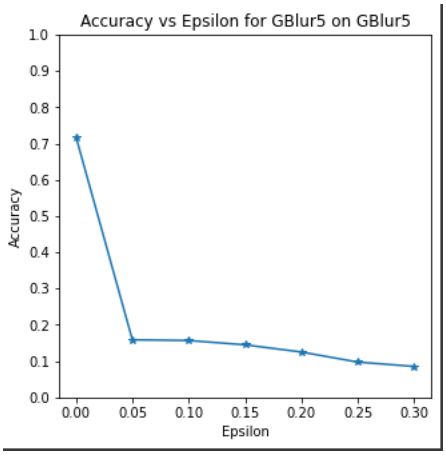
Blurring the victim model increases resilience against attacks up to a certain point (around 0.2 eps), but stronger attacks (>0.2) overcome the resilience

Test A2
Attacker model= GBlur5
Victim Model= Varied (Clean to blurred)
Input dataset= Same as attacker (GBlur5)







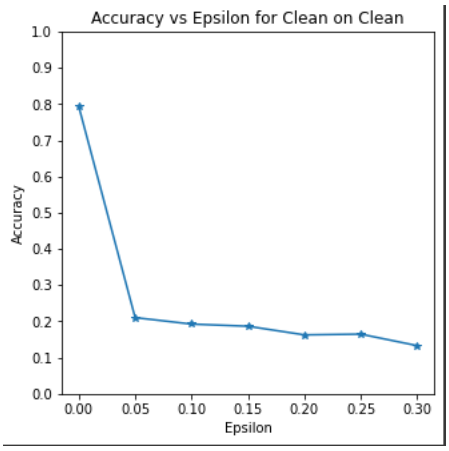


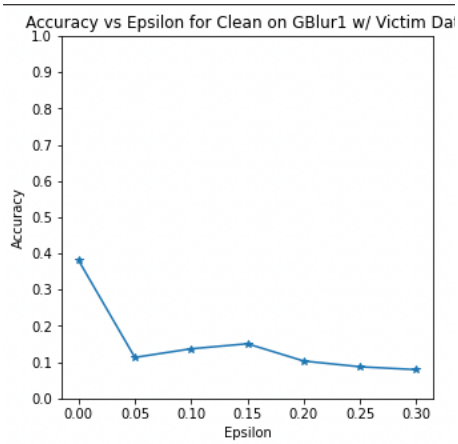
Attacks with epsilon 0 correctly show the victim model's accuracy increasing as its dataset gets closer to resembling GBlur5, since they are classifying images from GBlur5's dataset. The attack seems to be more consistent in decreasing the accuracy when carried out on the model it is intended to attack- GBlur5.

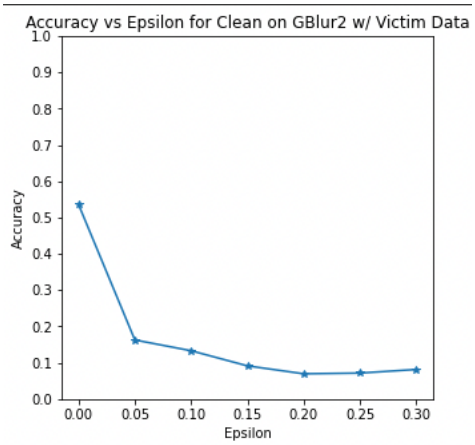
Test Set B: Victim Model's Dataset as Input

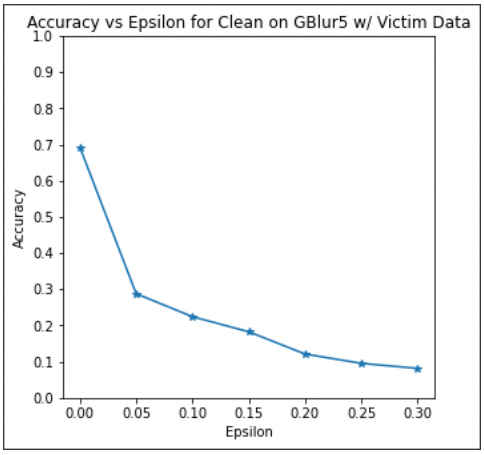
Test B1

Attacker model= Clean
Victim Model= Varied (Clean to blurred)
Input dataset= Same as victim (Varied)









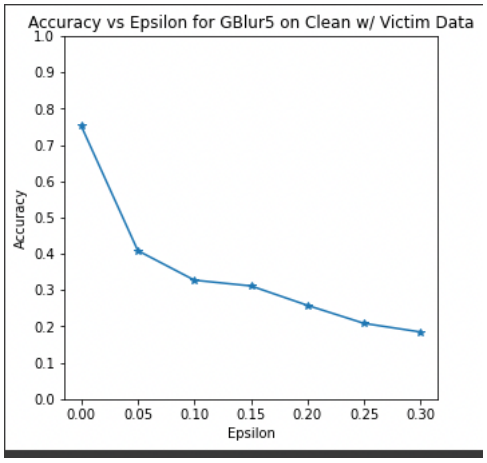
Attacks with epsilon 0 correctly show very varied accuracies. Some indication of possible ridge (.05-.1 in attack on gblur2) that there's robustness (resiliency against attack) against small attacks. There are strange behaviors that need further testing.

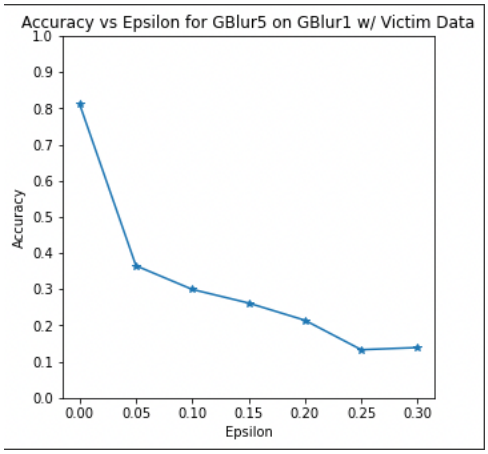
Test B2

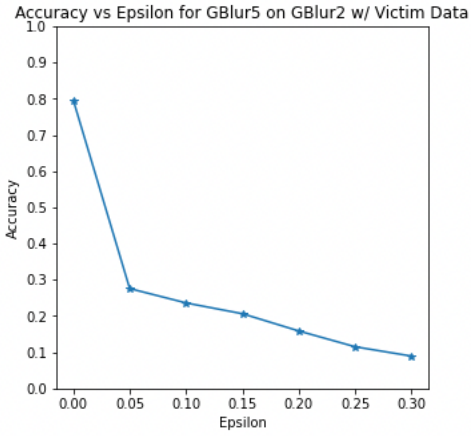
Attacker model= GBlur5

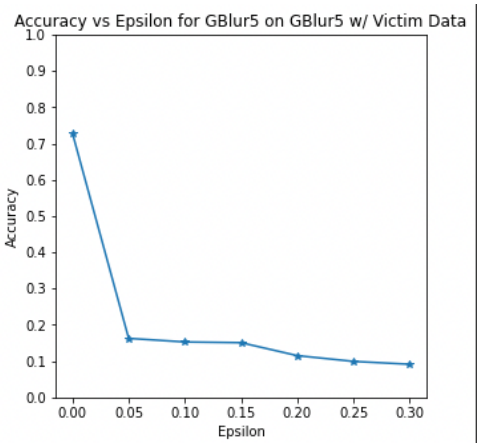
Victim Model= Varied (Clean to blurred)

Input dataset= Same as victim (Varied)









This test is most conclusive to showing the impact of datasets to an attack's transferability. As the victim models that are attacked are trained on datasets increasingly similar to GBlur5, attacks become more effective. For smaller attacks, victim models trained on datasets more dissimilar to GBlur5 show increased resiliency against the attack. Stronger attacks over a certain point (around epsilon value of 0.15-0.2) overcome the resiliency against the attack.

Conclusion

Overall, although these results are very preliminary, they lead to some conclusions. Results showed that **as the datasets the victim models were trained on became more dissimilar/blurrier than the dataset the attacker model was trained on, the victim model had more resiliency against the attack. However, adversarial attacks stronger than a certain threshold (between epsilon value of 0.15-0.2) overcame this resilience**, fooling all the victim models at similar levels.

In national security, this could imply that it is crucial for the US to not keep its datasets public. For instance, if Russia wanted to fool US image-classifying satellites to misclassify an image using adversarial examples, it is important that the datasets used to train Russia's attacks are most dissimilar to the US'. This can build resiliency/robustness for US models against Russian adversarial attacks.

However, this builds resiliency only until Russia decides to make their attacks stronger than a certain threshold (> epsilon value of 0.15-0.2). Attacks stronger than this point is likely to overcome US' resiliency and effectively fool its models. Yet, if attacks stronger than this threshold become more obvious to detect, the US may be rest assured that Russia would not generate attacks as strong.

A notable concern is that each time a saved model was loaded either as a victim or attacker model before a test, its accuracy would change while hovering between 75%-81%. Additionally, training models took a relatively long time, even when using transfer learning. Training a DenseNet model took nearly 6 hours. Transitioning to a virtual machine with stronger computing power may have mitigated this significantly.

Moving forward, more tests must be performed to ensure consistent findings and explain strange patterns.