Team name: "Deep Neural Networks, They Just (Don't) Work"
Team members (UWNetID is the same as CSE NetID):

- Kushal Jhunjhunwalla (kushaljh@cs)
- Andrew Wei (nowei@cs)
- Leiyi Zhang (leiyiz@cs)

Repo: https://github.com/nowei/adversarial-dl

Important sentences highlighted for brevity. If you just read the highlighted portions + pictures, it's between 2-4 pages. We finished writing this report last week, but the spec was updated in the meantime; and we'd hate to see our efforts go to waste, so we just recommend reading the highlighted portions.

Motivation

Deep Learning overthrew feature engineering in the task of Image Classification and now we have accurate models for large datasets like Image-Net. We wanted to show that these models weren't robust to some targeted attacks that adversaries could make. Specifically, we wanted to show that when we perturbed the image in a way that is hard for people to notice, we could obtain wildly inaccurate classifications. Our goal was to show people that deep learning still had a long way to go in terms of robustness. This is cool because it showcases how fragile these models are. As we are moving towards automation and use of deep NNs for classification tasks, these attacks prove fatal in critical situations like object detection via cameras in surveillance and self-driving cars.

Data Used

The model we used was trained on **ImageNet**. We test on any jpeg image.

Related Work

MadryLab's robustness: This is basically a library focused on adversarial training and example generation. They take pretrained models and pre-set data loaders for specific datasets and generate adversarial images from taking it from the data loader. So they don't attempt to create adversarial examples based on new data. This makes sense since one of the main functionalities of the library is to train robust models.

Advis.js: Visualization for FGSM that is pretty similar to what we did in terms of choosing the perturbation constant, but they we allow more customization of parameters and targeted attacks. They use TensorFlow and MobileNet instead of PyTorch and ResNet.

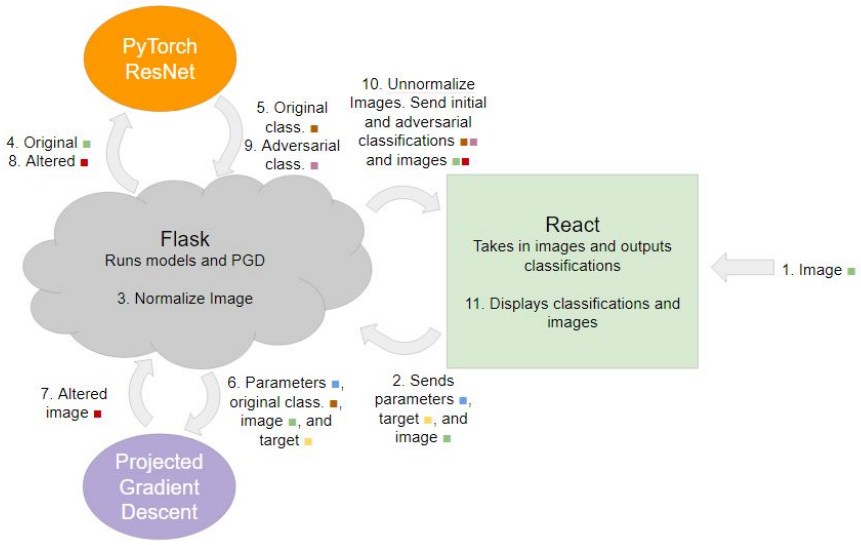
Components from pre-existing work

- Explaining and Harnessing Adversarial Examples (2014), Ian J. Goodfellow and Jonathon Shlens and Christian Szegedy
  - Where we first saw the panda as gibbon image and learned about adversarial training and FGSM.
- Towards Deep Learning Models Resistant to Adversarial Attacks (2017), Aleksander Madry and Aleksandar Makelov and Ludwig Schmidt and Dimitris Tsipras and Adrian Vladu
  - Where we learned about PGD and got an idea of how to code PGD.
- Know Your Enemy by Oscar Knagg - Medium article
  - They had an implementation for MNIST that we extended to work with what we had for ImageNet. We used this resource as the basis for our adversarial image generation.
- PyTorch ResNet Implementation
  - We took a ResNet-34 model from PyTorch for the base classification, the adversarial image generation, and attempts at adversarial training.
  - We did this because if we trained our own models, people could claim that the model might've just been terrible to begin with. We wanted to show that state-of-the-art models aren't robust rather than some model that we hacked up over a few weeks wasn't robust.

Components we implemented for the project

We ended up making the React front-end in JavaScript, figured out a way to pass the images we upload to the Flask Python backend by using a byte-stream, hooked that up with PyTorch, then passed the images back. We originally had everything on a jupyter notebook, which we ended up porting to Flask. We figured out how to undo the normalization to get the images back in a human-readable state.

We decided to use more existing resources rather than trying to reinvent the wheel, which we felt was a better use of our time and easier to defend if someone claimed that our models might've just been bad to start with.



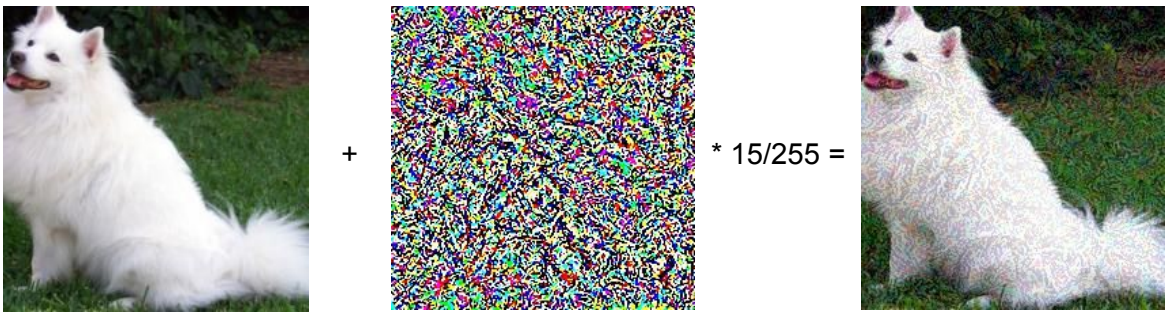
Pipeline of adversarial perturbation process

Techniques

We originally considered just using random noise to perturb the image and scaling it down so that people could not notice it. When we tried this, we saw that it didn't work. Then we talked to Aaron about it and learned that most models are robust to random noise anyways, so we'd have to do some targeted attacks on the image we're classifying. We had read some things about this when reading the papers, but it never clicked for as to why we had to do this until then. We wanted to perturb the image in a way such that it was close to the original image, but moved into a space that would no longer be classified as the original class.

Attempt with random Gaussian noise

We then considered using **Fast Gradient Sign Method (FGSM)** and **Projected Gradient Descent (PGD)**. FGSM is essentially taking the gradient, and moving in the direction opposite the gradient by some perturbation factor, epsilon. FGSM ended up perturbing it too much for some images and was generally inconsistent in how much we had to perturb the images to get an alternate classification. We were also not able to recreate the panda classified as gibbon image using this technique, which was what the paper said it used, but we chalk this up to the paper using a different model than what we used (they used GoogLeNet and we used ResNet).



Samoyed, 89.36% + prayer rug, prayer mat, 16.02% * ε = poncho 33.40%
Attempt with FGSM

Then we looked into Projected Gradient Descent. This ended up working pretty well. Projected Gradient Descent is essentially doing the same thing as the FGSM in the sense that it moved against the gradient (or towards it if we have a target classification), but then bounding the error on each pixel to plus or minus epsilon so we don't perturb each pixel by the same amount (essentially projecting onto $L_{infinity}$ ball [maximum epsilon difference in any pixel]).

1. Eskimo dog, husky **53.94%**
2. Siberian husky **25.52%**
3. malamute, malemute, Alaskan malamute **20.03%**
4. dogsled, dog sled, dog sleigh **0.18%**
5. Norwegian elkhound, elkhound **0.11%**

1. toilet tissue, toilet paper, bathroom tissue **39.02%**
2. paper towel **11.98%**
3. Siberian husky **6.58%**
4. Eskimo dog, husky **5.08%**
5. Border collie **2.72%**

Attempt with PGD

We decided to display the top-5 classification instead of the top-1 classification because we wanted to make sure that a relatively significant change was induced by the perturbations rather than a small one, like the correct classification didn't just move down by a few percent.

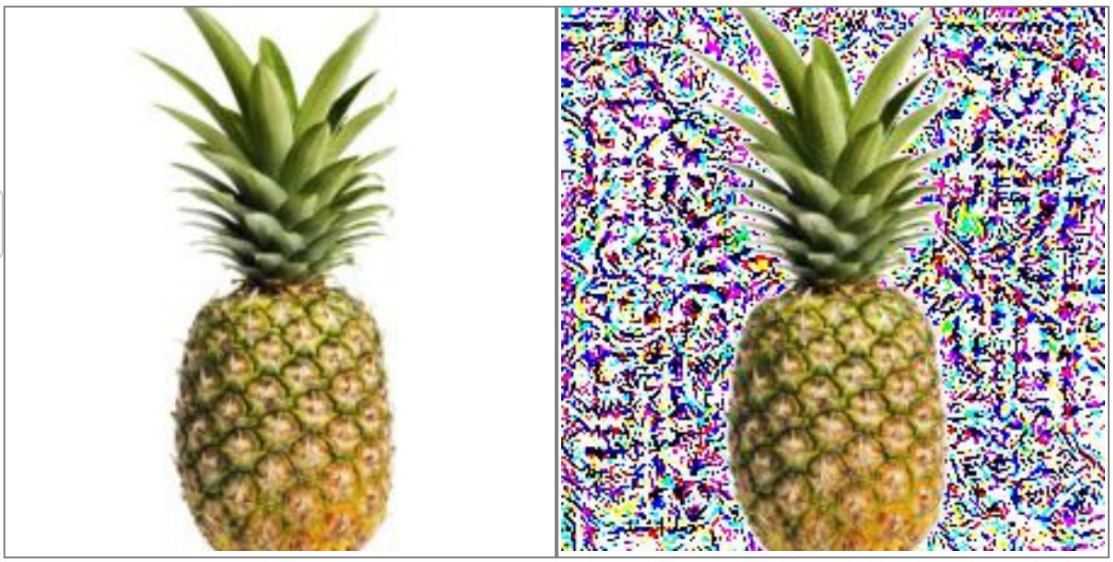
Experiments

The experiments we ran were mostly to test how the system reacts when we try different images and targets. The left is the original classification. The right is the perturbed image.

1. giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca **99.97%**
2. indri, indris, Indri indri, Indri brevicaudatus **0.01%**
3. gibbon, Hylobates lar **0.00%**
4. American black bear, black bear, Ursus americanus, Euarctos americanus **0.00%**
5. colobus, colobus monkey **0.00%**

1. gibbon, Hylobates lar **97.70%**
2. siamang, Hylobates syndactylus, Symphalangus syndactylus **1.09%**
3. indri, indris, Indri indri, Indri brevicaudatus **0.36%**
4. giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca **0.29%**
5. langur **0.13%**

Here, we recreated the classic image and have a panda classified as a gibbon. We see that gibbon appears to be one of the top 5 classifications without perturbing it, albeit it sits at 0% due to the rounding. Apparently gibbons come in many colors and some have white and black fur, so it is actually somewhat believable that a panda would be misclassified as gibbon. After we perturbed the image, the network classified it as a gibbon with a high confidence. Then the next two are still primates and the 4th is just the panda again, but we see that the confidence in which it believed it was a panda dropped significantly.



1. pineapple, ananas **100.00%**
2. strawberry **0.00%**
3. banana **0.00%**
4. custard apple **0.00%**
5. daisy **0.00%**

1. fountain pen **98.82%**
2. ballpoint, ballpoint pen, ballpen, Biro **0.42%**
3. rubber eraser, rubber, pencil eraser **0.11%**
4. quill, quill pen **0.09%**
5. crossword puzzle, crossword **0.08%**

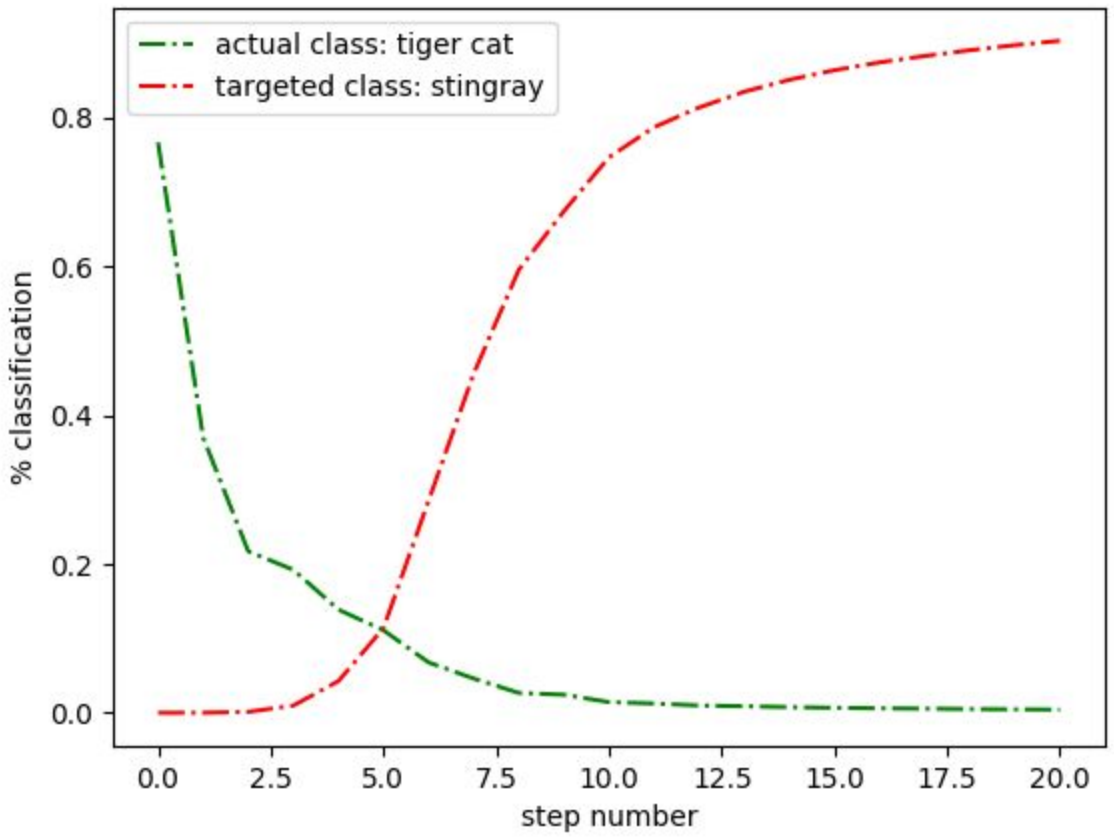
We wanted to see if we could make a pineapple into a pen, the network is so confident that this pineapple is a pineapple that we had to crank up the epsilon and step size. This is an interesting result because this ended up perturbing everything except for the pineapple. We suspect that this image was inside the training data, as it's pretty confident that it's a pineapple. The other explanation for the same could be that because of a white background, there is nothing else that the network can consider the image to be. Notice that the initial classifications were other fruit, which might give us a better idea of where the pineapple existed relative to other classes, i.e. it lived in some sort of a "fruit" space.

1. gibbon, Hylobates lar **97.43%**
2. titi, titi monkey **0.90%**
3. patas, hussar monkey, Erythrocebus patas **0.50%**
4. indri, indris, Indri indri, Indri brevicaudatus **0.31%**
5. siamang, Hylobates syndactylus, Symphalangus syndactylus **0.25%**

1. gibbon, Hylobates lar **99.34%**
2. titi, titi monkey **0.18%**
3. siamang, Hylobates syndactylus, Symphalangus syndactylus **0.11%**
4. patas, hussar monkey, Erythrocebus patas **0.11%**
5. indri, indris, Indri indri, Indri brevicaudatus **0.09%**

We wanted to see if we could make a gibbon more gibbon-like and what that would entail. The more general statement is we wanted to see if we can enhance the correct result and we did. We essentially made it more confident that it was a gibbon instead of another type of monkey.

We made an analysis of the above cat image (left) and the perturbed version of it (right).



The resultant graph above depicts 20 steps of PGD using the cat picture above and with a stingray chosen as the target. We see that original probability goes down and the target probability goes up.

Conclusions

We are generally happy with how this project turned out, since it is relatively demonstrable that when we perturb the images slightly, the classifications can become wildly incorrect for a modern neural network. We often hear about how great deep learning models have gotten, and every year there are new research papers about increasing the performance of these models on some tasks; but there seems to be a lot less papers on how to make these models more robust when it encounters people or programs that try to trick them. We believe this is one of the fundamental roadblocks in deploying deep learning systems to fields that requires and prioritizes correctness and security, since if someone manages to fool the system in a mission-critical setting like a self driving car, it could have irrevocable consequences. We hope to remind people that while deep learning is great, we still have a long way to go before it becomes the amazing tool that we all envision it to be.

APPENDIX

Problems Encountered

- **Getting ImageNet data**: ImageNet no longer available through PyTorch because ImageNet didn't own the copyrights to the images, so now it's only available to researchers and educators, but when we requested it a few weeks ago, we never got a response when the website said it would respond within 5 business days.
- **FGSM perturbed it too much**: Whenever we used FGSM, it ended up perturbing the image too much and the noise was visible, which is something that we wanted to avoid for the sake of our project because the point was to show that small, imperceptible perturbations could alter the image classification.
- **Training a robust model**: Since training on ImageNet is computationally intense we didn't have access to enough compute or have the time to fully train a model; we wanted to show that adversarial training for one class would make the model more robust for classifying that class. We tried and eventually realized that continuing training for one class heavily biased that one class and would ruin the weights for all other classes.
- **Perturbations between pixel values get truncated when saving the image if the perturbations are too small**: This causes the model to classify the image in the same way as the original image when we saved it and tried classifying that or maybe the classification was only slightly different from the original image. We believe this is what happened for several papers and repositories that provide similar setups, but don't consider this possibility.
- **Normalization**: The ResNet model we used was trained on normalized images, so to be accurate and consistent, we had to work with normalized images, perturb them slightly, classify it, then undo the normalization to display them. This was just a problem we had to work with, but we added the perturbations and then scale it back the same way, so the normalization shouldn't have been what affected the classifications.

Future Work

- Figuring out a way to save the images such that we can store the small perturbations, which we can probably do by using an alternative image format.
- Robust model to more than just a single class.
- Possibly hosting this on an accessible website so that people can test it out themselves in their free time. This currently needs to be hosted locally by running both React and Flask.
- Porting this over to a docker container (if they don't go out of business by then) so that it's easier to host
- Allowing for different types of changes like rotations, resizing, or adding a random pixel in different places. These are all inspired by different attacks we've seen while deciding on which attack we should focus on.
- Testing this out on a more state-of-the-art model, like EfficientNet or FixResNeXt, and make an adversarial model that generates images that are network agnostic.