ADVERSARIAL EXAMPLES FOR NEURAL NETWORKS

JUSTIN WONG

Contents

Date: May 16, 2019.

1. Abstract

Neural networks have a wide range of applications ad have achieved great results in many of these tasks. However, neural networks are not without weaknesses. One such weakness is a susceptibility to adversarial attacks. That is, it is possible to manipulate inputs to the neural network with small changes in such a way that will cause it to return faulty results. This project looks to explore possible attack vectors and vulnerabilities that networks may be vulnerable to and see how effective these attacks are.

2. Introduction

Neural networks have seen a surge in usage for many different applications ranging from classification tasks to recognition tasks to detection tasks to synthesis tasks. It has accomplished feats in fields ranging from finance, computer vision, and games and has seen great success with the introduction of huge aggregated datasets to train on. These can either be publicly available or gathered by commercial means. However, the amount of success that it has had does not signal that it has no weaknesses. Neural networks are well known as a type of black box solution to problems. While it may generate great outputs for given inputs, the methods at which it arrived at the output would not be known. That is, one can verify that the output is correct, but finding the exact "reasoning" why the neural network arrived at that conclusion would not be well described (of course the architecture and machinery is well understood but the subcomponents of the problem that may allude to the solution in similar cases will not be very clear). While this may not be a problem in many domains, it definitely has implications on a wide spectrum of others. One such example of the problem of the black box reasoning of neural networks is the existence of adversarial examples. A vulnerability to adversarial examples leads to the ability for malicious entities to create examples that look like other normal inputs, but are classified very differently by the neural network.

These results can be seen to be wrong with the naked eye, but the neural network still misclassifies them. While the exact reason is not known, a possible explanation is that the network suffers from some form of underfitting. More specifically, it is suggested that the adversarial examples play at the decision boundaries of the different class that the network is trained on. If the network was able to create exact boundaries at each of the classification classes that perfectly encompass every example of each class, then there would be no such attack available. Since such perfect boundaries are not possible, there should be some regions around every class boundaries where there can be misclassification. Moreover, it is around these boundary positions are the closest to the original input and the easiest to find using gradient methods. The existence of such an attack vector is important because vulnerabilities to adversarial examples are major problems in many accuracy sensitive and safety critical tasks. Examples include the development of self-driving cars and other applications which likely will not have public trust unless adversarial examples are accounted for in the training systems.

In this project, we explore the attack methods of Fast Gradient Sign Method (FGSM) and Iterative Method (IM) in both the untargetted attacks and untargetted attacks. Adversarial examples can be seen as generated inputs that have the same core essence as the original input but fool the neural network into misclassifying it. Specifically, the original input and the adversarial example should obviously fall under the same class to any human observer, but cause the neural network to classify them differently. Untargetted attacks simply look to create an adversarial example that causes the neural network to misclassify the input. In this case, what the neural network misclassifies the adversarial example as is irrelevant so long as it is not the intended classification. On the other hand, the targetted attack looks to choose a class that the neural network should misclassify the adversarial example as. In this case, an attacker should be able to cause the neural network to classify the input as a different class if their choice.

2.1. **Related Work.**

The Carlini and Wagner attacks (CW) will not be explored in this project, but are another method of attack that has been recently developed. This is a more modern attack that looks to find an adversarial example while minimizing the distance from the original input with respect to some distance metric. Currently, the authors have provided code samples that demonstrate this for the L_1, L_2, L_∞ norms, though optimization for any distance metrics are possible since the optimization is simple a gradient descent method executed for the loss over the distance metric. Let $x \in \mathbb{S}$ be the original input classified under the network N as $N(x) = l$. Then CW looks to create an adversarial example x' close to x under some distance metric $d(\cdot, \cdot)$ that satisfies

$$\begin{aligned} \min_{x'} \quad & d(x, x') \\ s.t. \quad & N(x') \neq l \\ & x' \in \mathbb{S} \end{aligned}$$

Of course, that is the untargetted formulation. For a target class l', the formulation becomes

$$\begin{aligned} \min_{x'} \quad & d(x, x') \\ s.t. \quad & N(x') = l' \\ & x' \in \mathbb{S} \end{aligned}$$

However, this method takes the longest though it does guarantee a result on all current neural networks with one hundred percent probability [3]. Implementation of this attack is another possible advancement of this project.

3. Methods

3.1. **Data.** To begin, we choose to utilize Google Colaboratory in order to make use of the GPU computing resource to improve performance and conserve time. As such, there are additional utilities that need to be added, specifically including a method for uploading local files into the virtual environment. This can be seen in the appendix.

Since the network being used is pretrained on the ImageNet dataset, it has outputs that return a probability of the input being under each of the classes. Thus, the images that are used for the experiment can be any image, but should ideally be images that fall under at least one of the classes in ImageNet. There are 1000 possibilities and the full catalog of classes can be found under $https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a$.

3.2. **Model.** We utilize Keras and tensorflow for the neural network and backend and use ResNet50 that has been pretrained on ImageNet that is available in Keras. We also utilize Keras functions for processing images and matplotlib to plot results. Time is used for simple timing of results that will be discussed later.

ResNet50 is used since it is a strong model trained on the ImageNet dataset that is also small in size. This is important since the high original accuracy will allow for better observations to be made on the effects of the adversarial examples on the network. The smaller size is also important since the computation of gradients will also take a shorter amount of time, allowing for more computations to be made (especially since the GPU resource is limited even with the usage of Google Colaboratory). Lastly, the residual blocks in ResNet also help prevent the network from having vanishing gradients which will also allow for the FGSM methods to better find non zero local gradients to find steps towards adversarial examples.

Listing 1. Python packages utilized.

```python
import numpy as np
from keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
from keras.preprocessing import image
import keras.backend as K
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
import time
```

3.3. **Methodology.** There are two flavors of FGSM: targetted and untargetted. The idea for the untargetted attack is to take the local gradient and move in a direction that is opposite from the descent direction. Let $x_{original}$ be the input, x_{adv} be the adversarial output, and $O(x, y)$ be the objective function. Then this can be expressed as

$$x_{adv} = x + \epsilon sign(\nabla(O(x, y_{original}))$$

.

Recalling that gradient descent step in the negative direction of the gradient, this can be seen as a single step gradient ascent with step size varying with choice of ϵ.

The idea for the targetted attack is instead of stepping away from the original class label explicitly, this is done implicitly by stepping towards a different class label. Keeping the same notation, this can be expressed as

$$x_{adv} = x - \epsilon sign(\nabla(O(x, y_{target}))$$

.

Unlike the untargetted attack, we are stepping in the negative direction of the gradient. This time, the attack is adding an auxiliary gradient descent step in the direction of an incorrect class label with step size varying with choice of ϵ [1].

It can be seen that the implementations of the two flavors of FGSM are very similar up to a sign change and the calculation of the gradient with respect to a class label. It is then simpler to create a single method that can be used to calculate both up to a change in the parameters passed.

Listing 2. Python function for generic FGSM for targetted and untargetted attacks.

```
def fgsm(model, img, target, targetted, epsilon):
    loss = K.categorical_crossentropy(K.one_hot(target, 1000), model.output)
    gradient = K.gradients(loss, model.input)

    signs = K.sign(gradient[0])
    if targetted:
      x_adv = img - epsilon * signs
    else:
      x_adv = img + epsilon * signs
    x_adv = sess.run(x_adv, feed_dict={model.input:img})
    return x_adv
```

Here, the function takes in the model being attacked, the original input, the class label, a boolean value describing the attack flavor, and the ϵ step size. The loss is set to the the cross entropy loss and the signs of the gradients are calculated with respect to the input and the class label provided. Notice that in the untargetted attack case, the class label is the original class and in the targetted attack case, the class label is the targetted class. Depending on the boolean, we perform either a single step ascent or descent and return the resulting adversarial image. This is one step of the FGSM method.

Listing 3. Python script for executing FGSM untargetted attack.

```
preds = model.predict(x)

initial_class = np.argmax(preds)
```

```
print('Predicted:', decode_predictions(preds, top=3)[0])
epsilon_space = np.logspace(-2, 0, num=40)
for e in epsilon_space:
  x_adv = fgsm(model = model, img = x, target = initial_class, targetted = False, epsilon = e)

  preds = model.predict(x_adv)
  dec = decode_predictions(preds, top=3)[0]
  print('Predicted:', dec)
```

LISTING 4. Python script for executing FGSM targetted attack.

```
preds = model.predict(x)

initial_class = np.argmax(preds)
target_class = 346
print('Predicted:', decode_predictions(preds, top=3)[0])
epsilon_space = np.logspace(-2, 0, num=40)
for e in epsilon_space:
  x_adv = fgsm(model = model, img = x, target = target_class, targetted = True, epsilon = e)

  preds = model.predict(x_adv)
  dec = decode_predictions(preds, top=3)[0]
  print('Predicted:', dec)
```

We check a variety of ϵ values separated on a log scale because the smaller step sizes are likely the ones that have higher relevance. Larger step sizes do not tell as accurate a story since direction may not be the best choice non locally. Notice that the main difference in the two scripts is the targetted boolean and the target.

Also, there is an increase in time spent per FGSM iteration as the model is used more. These graphs can be seen in the python notebook, but without resetting the model and the computational graph, there is significant slowdown as the model is used repetitively. Thus, we implement a model reset when the time taken for some iteration batch exceeds a set amount of time. This is especially useful for IM since we are performing many more FGSM calls and reduces the amount needed significantly.

There are also targetted and untargetted attack flavors for IM. The idea for the untargetted attack is to take the local gradient and move in a direction that is opposite from the descent direction. Unlike FGSM, it repeats this process using the generated example from the previous iteration as the new starting point. That is, using the output as the new input, a new local gradient is calculated and another step in the ascent direction is take. Using the prior notation, this can be expressed as

$$x_{adv}^{N} = x_{adv}^{N-1} + \epsilon sign(\nabla(O(x_{adv}^{N-1}, y_{original}))$$

$$x_{adv}^{0} = x_{original}$$

.

Notice that IM is essentially repeated FGSM iterations with updated inputs. This allows for the generation of adversarial examples to take many steps with updated local gradients such that the direction is always close to optimal. In essence, it allows for the adversarial example to get further away from the original example without using large step size (which would not generalize from the local gradient) and instead use iterative updates that mirror gradient ascent, best using local gradients.

Targetted IM is also similar to repeated iterations of targetted FGSM. Once again, the inputs are updated with every step and a new local gradient is used for each step. Using the prior notation, this can be expressed as

$$x^N_{adv} = x^{N-1}_{adv} - \epsilon sign(\nabla(O(x^{N-1}_{adv}, y_{target}))$$
$$x^0_{adv} = x_{original}$$

.

Again, we are stepping in the negative direction of the gradient every iteration. This makes it a case of gradient descent in the direction of the target class [2].

Listing 5. Python script for executing IM untargetted attack.

```
preds = model.predict(x)
initial_class = np.argmax(preds)
x_adv = x
time1 = time.time(), time.time()
for e in range(100):
    x_adv = fgsm(model = model, img = x_adv, target = initial_class, targetted = False, epsilon =
        0.01)
    if e%5==0:
      preds = model.predict(x_adv)
      dec = decode_predictions(preds, top=3)[0]
      print(e, 'Predicted:', dec)
      diff = time.time() - time1
      print('previous 5 iteration of fgsm took: {} seconds'.format(diff))
      time1 = time.time()
    if diff > 45:
      reload_model()
      diff = 0
```

Listing 6. Python script for executing IM targetted attack.

```
initial_class = np.argmax(preds)
target_class = 346
x_adv = x
time1 = time.time(), time.time()
for e in range(100):
    x_adv = fgsm(model = model, img = x_adv, target = target_class, targetted = True, epsilon =
        0.01)
    if e%5==0:
      preds = model.predict(x_adv)
      dec = decode_predictions(preds, top=3)[0]
      print(e, 'Predicted:', dec)
      diff = time.time() - time1
      print('previous 5 iteration of fgsm took: {} seconds'.format(diff))
      time1 = time.time()
    if diff > 45:
      reload_model()
      diff = 0
```

Again, the main difference between the targetted attack and the untargetted attack for IM is the target and the targetted boolean. A small ϵ value is chosen since we want to take small steps in line with the local gradient. Furthermore, there is an additional difference since the source input of IM is being updated with every FGSM iteration. Specifically, the adversarial output of the previous call is the input of the next call. In this way, a new local gradient is calculated at each point and the direction will be updated as the example traverses the landscape. There is also an added utility to check if the time taken per iteration batch exceeds a certain time so that the model can be reloaded. This saves time since over many FGSM iterations, computation becomes slow and a reloading of the model is necessary to reset computation times.

3.4. **Procedure.** For the FGSM results, we verify that the method works for a variety of inputs that fit the input space of ImageNet classes. For sake of space, we include only the results of a single image. However, one can verify that the method works on other images in the input space using the code samples or the python notebook file uploaded.

The testing is carried out by performing a single FGSM step on the input sample under many different ϵ step size values. We generate results that reflect the behavior and properties of the adversarial example under each of those conditions and demonstrate them below. A similar method is carried out for both the targetted and the untargetted flavors. We additionally test some different classes for the targetted examples and, again for the sake of brevity of the report, include one with the best results.

For the IM results, we verify that the method works for a variety of inputs that fit the input space of ImageNet classes. For sake of space, we include only the results of a single image. However, one can verify that the method works on other images in the input space using the code samples or the python notebook file uploaded.

The testing is carried out by performing multiple FGSM steps on the input sample tested under a few ϵ step size values. Choosing the one with the best results, we extend the iteration count so that the behavior of the adversarial examples under the attack can be observed with more detail. The results are saved in batches of 5 epochs and the results are generated from the properties of the adversarial examples in the saved array. A similar method is carried out for both the targetted and the untargetted flavors. We utilize the same class found in the FGSM targetted example for the IM targetted example.

4. Results

We explore the results of different adversarial attacks on a pretrained network. In this paper, we utilize the ResNet50 network pretrained on the ImageNet classes that is available in Keras.applications. Any image will work for the attack. For these results, we will mainly talk with respect to this chosen image of a sea lion.

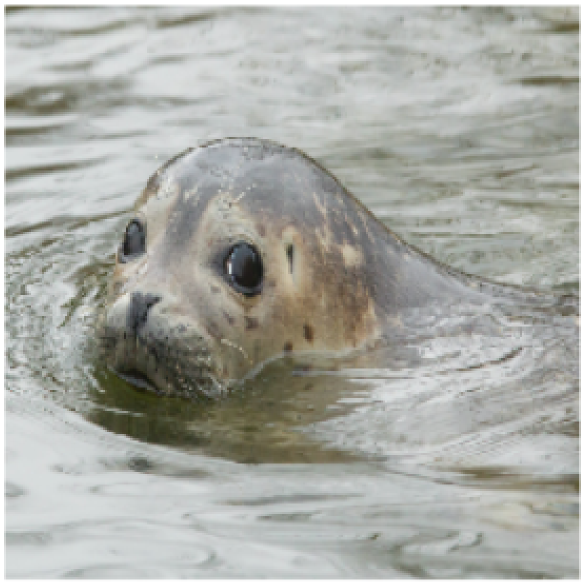
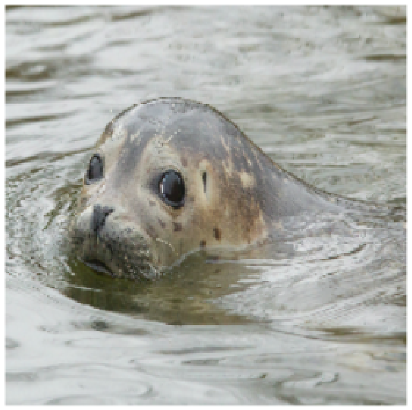
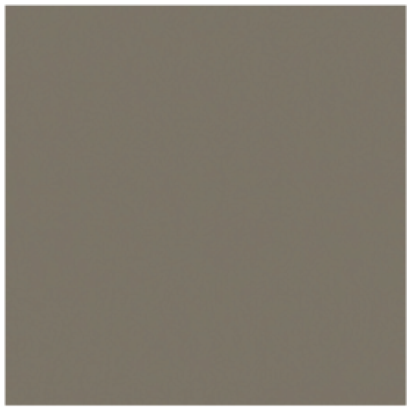


FIGURE 1. Original image of a sea lion. Under ResNet50, its top three class predictions are sea lion with confidence 0.91135, leatherback turtle with confidence 0.061980836, and otter with confidence 0.008506746.

We first explore the results of some untargetted FGSM attacks.

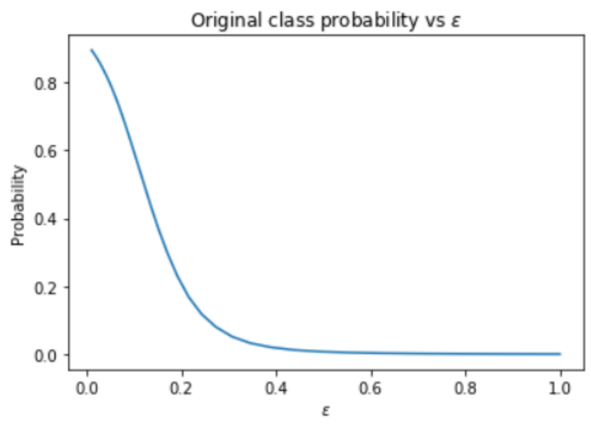


(A) Adversarial example generated from the original image using untargetted FGSM on ResNet50 with $\epsilon = 1$. This is classified as a leatherback turtle with confidence 0.850928.

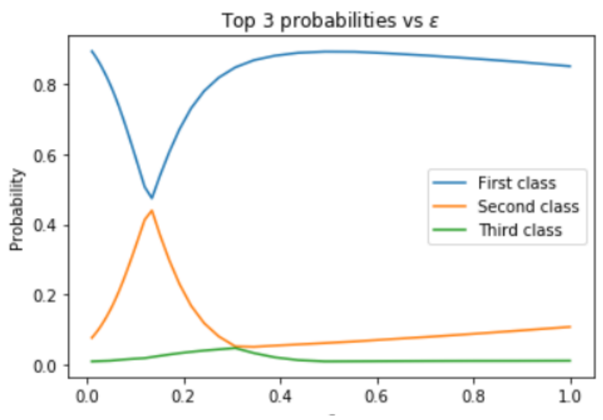


(B) Pixel difference of the generated adversarial example in (A) with the original image. This is classified as a kite with confidence 0.07894931.

FIGURE 2. Visual results of untargetted FGSM attack on ResNet50 on $\epsilon = 1$.



(A) ResNet50's prediction of the probability of the adversarial example being the originally classified class varying with respect to varying ϵ.



(B) ResNet50's probabilities of the top three classes of the adversarial example varying with respect to varying ϵ.

FIGURE 3. Variation of the confidence of ResNet50 in its top three predictions and of the original class as a function of ϵ.

Notice that the confidence for the original class can be traced as along the confidences of the top three classes. Specifically, notice that the original class is no longer the highest probability class a little before $\epsilon = 0.2$ and continues to decrease with increasing ϵ. By $\epsilon = 1$, the confidence of the original class has reached very nearly 0. Meanwhile, the other classes that are being predicted have reached confidence levels that rival the initial confidence of the original class of the non -adversarial example.

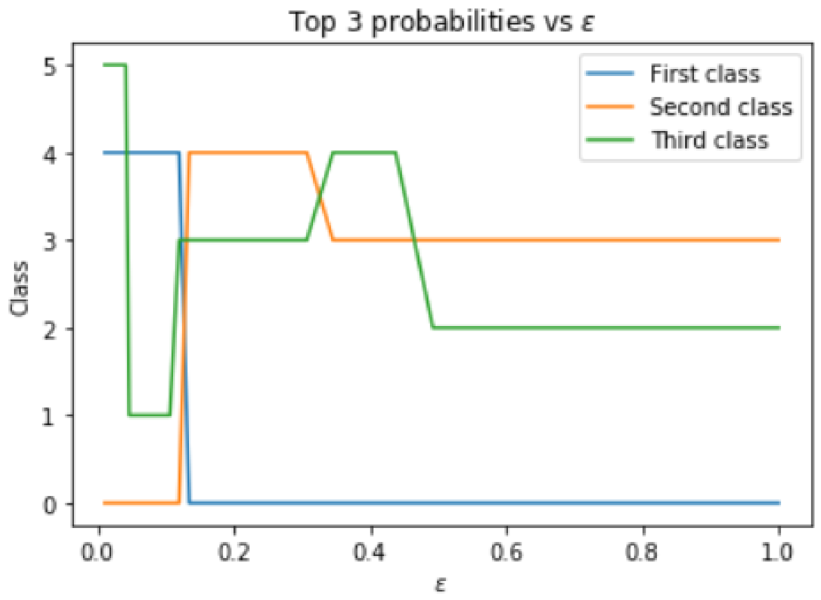


FIGURE 4. ResNet50's top three class predictions adversarial example varying with respect to varying ϵ.

There are six different classes that appear either as the original classification or inside the top three class predictions for the adversarial model. They are as follows:

- 0: Leatherback Turtle
- 1: Hippopotamus
- 2: Stingray
- 3: Loggerhead
- 4: Sea Lion
- 5: Otter

Notice that the initial top class is the sea lion. However, as seen in the probabilities graph, the ResNet50 no longer has the highest confidence in the class before $\epsilon = 0.2$. It then has some appearance as the second and third most probable classes before it finally drops out of the top three predictions around $\epsilon = 0.5$. However, as can be seen in the probabilities graphs, the confidence in the original class is not much less than the confidence in the third most probable class. In fact, the probabilities are very small because the confidences in the top two classes are so large. That is, the adversarial example has made ResNet50 classify with great certainty an incorrect label using only small perturbations on the original image.

It is also worth noting that the result that ends up being the most probable class is the leatherback turtle, which was the second most probable class on the original image. This makes sense because in FGSM, the adversarial example is simply trying to move in the opposite direction of the downwards gradient of the sea lion. In that case, it is not unlikely that a nearby class is the direction in which it is moving. Moreover, being the second most probable class also suggests that it may be closer to the decision boundary of the sea lion than other classes. As such, it has become the most likely class as the adversarial example moves further from the original class.

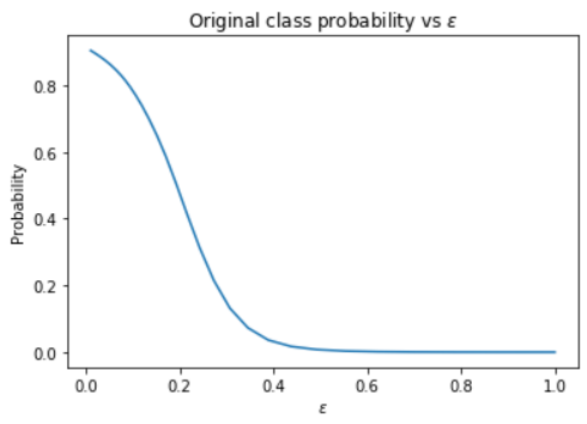
It can also be seen that the confidence in the highest probability class begins to decrease again after ϵ reaches a certain magnitude. This can easily be seen as a result of the global landscape no longer reflecting the local gradient. While the direction holds up well for smaller ϵ, the trend does not continue to generalize for larger perturbation.

We now explore the results of some targetted FGSM attacks. We set the target to class 346 in ImageNet or the water buffalo.

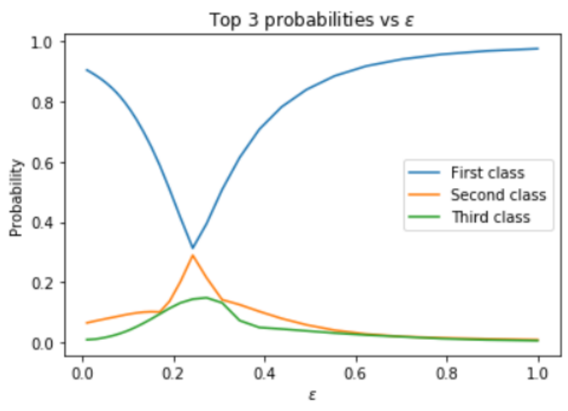
(A) Adversarial example generated from the original image using targetted FGSM on ResNet50 with $\epsilon = 1$. This is classified as a water buffalo with confidence 0.9746216.

(B) Pixel difference of the generated adversarial example in (A) with the original image. This is classified as a kite with confidence 0.06943793.

FIGURE 5. Visual results of targetted FGSM attack on ResNet50 on $\epsilon = 1$.



(A) ResNet50's prediction of the probability of the adversarial example being the originally classified class varying with respect to varying ϵ.



(B) ResNet50's probabilities of the top three classes of the adversarial example varying with respect to varying ϵ.

FIGURE 6. Variation of the confidence of ResNet50 in its top three predictions and of the original class as a function of ϵ.

Notice that the confidence for the original class can be traced as along the confidences of the top three classes. Here, the original class is no longer the highest probability class a little after $\epsilon = 0.2$ and continues to decrease with increasing ϵ. By $\epsilon = 1$, the confidence of the original class has reached very nearly 0. It is reasonable that it takes a larger ϵ to change the most probable class since the adversarial example is moving towards a different classification instead specifically away from the original one. Also notice that the second and third probabilities (as well as the ones under it) converge to very small. It can be seen that with higher ϵ, the confidence of non targetted classes become near negligible. In this case, at $\epsilon = 1$, we get probabilities less than 0.01 for the second most likely class.

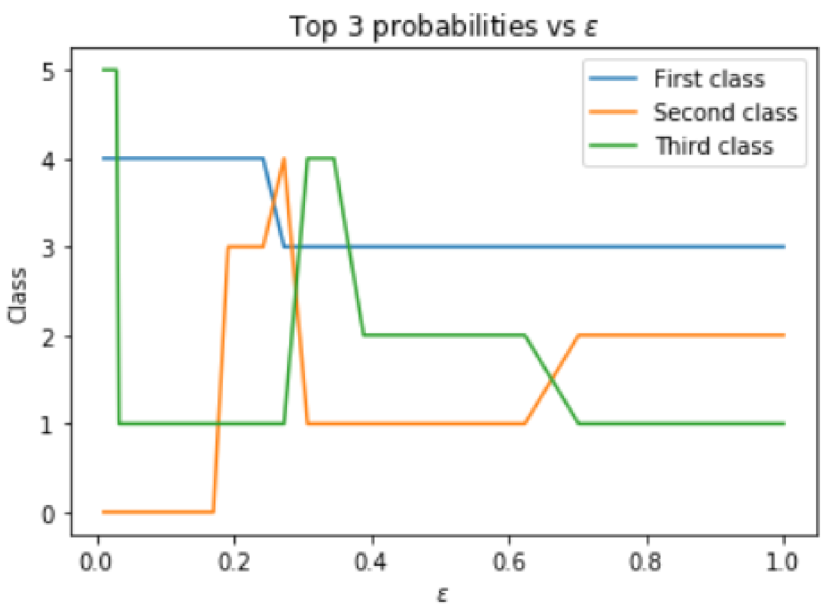


FIGURE 7. ResNet50's top three class predictions adversarial example varying with respect to varying ϵ.

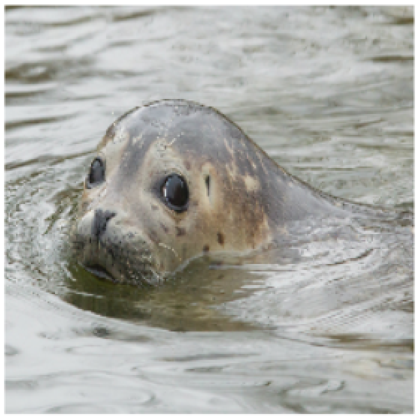
There are six different classes that appear either as the original classification or inside the top three class predictions for the adversarial model. They are as follows:

- 0: Leatherback Turtle
- 1: Hippopotamus
- 2: Indian Elephant
- 3: Water Buffalo
- 4: Sea Lion
- 5: Otter

Notice that the initial top class is the sea lion. However, as seen in the probabilities graph, the ResNet50 no longer has the highest confidence in the class after $\epsilon = 0.2$. It then has some appearance as the second and third most probable classes before it finally drops out of the top three predictions around $\epsilon = 0.4$. While it does exit the top three predictions faster than the untargetted attack, this can be explained since the local gradient does not capture the global landscape. It simply turns out that this direction chosen gets the the decision boundary faster despite it not being the exact same as the opposite of the local gradient.

While the most probable class became the targetted class as desired, notice that other classes of significant confidence remain similar and the classes that show up also end up being in the top possibilities are the same. This is likely due to the direction of change being determined locally. Then there are not many options that will get far away from all the decision boundaries.

We now explore the results of some untargetted IM attacks.


(A) Adversarial example generated from the original image using untargetted IM on ResNet50 with 100 iterations on $\epsilon = 0.01$. This is classified as a leatherback turtle with confidence 0.9844657.

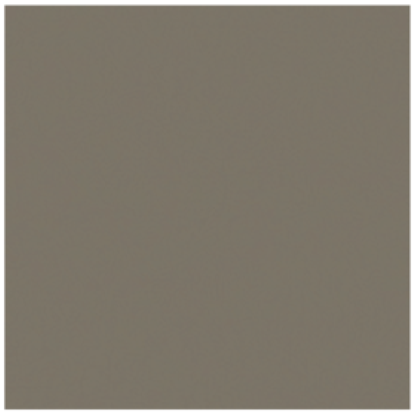
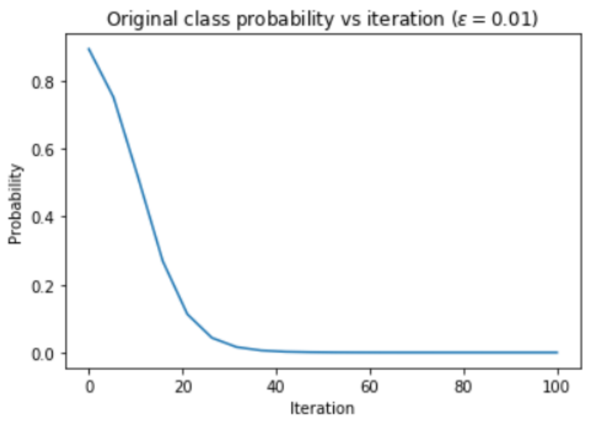
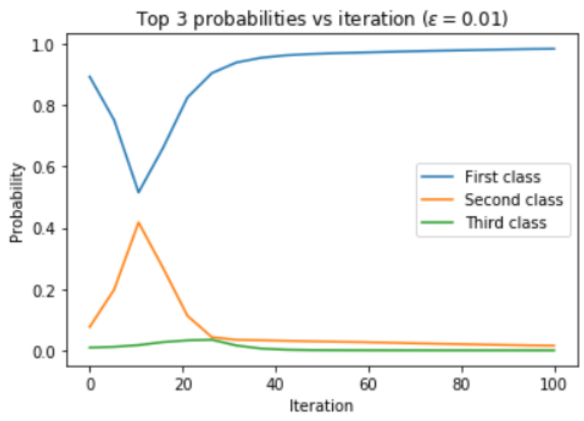

(B) Pixel difference of the generated adversarial example in (A) with the original image. This is classified as a space shuttle with confidence 0.068613775.

FIGURE 8. Visual results of untargetted IM attack on ResNet50 with 100 iterations on $\epsilon = 0.01$.



(A) ResNet50's prediction of the probability of the adversarial example being the originally classified class varying with respect to varying iteration.



(B) ResNet50's probabilities of the top three classes of the adversarial example varying with respect to varying iteration.

FIGURE 9. Variation of the confidence of ResNet50 in its top three predictions and of the original class as a function of iteration count.

Notice that the confidence for the original class can be traced as along the confidences of the top three classes. Specifically, notice that the original class is no longer the highest probability class a little before iteration 15 and continues to decrease as more iterations pass. By iteration 40, the confidence of the original class has reached very nearly 0. Meanwhile, the other classes that are being predicted have reached confidence levels that rival the initial confidence of the original class of the non-adversarial example. There is also a subtle difference in the speed at which the probabilities change. Specifically, for the original class IM causes a faster decrease with iteration that does FGSM with increasing ϵ. This is reasonable since IM takes a smaller step with updated local gradients whereas FGSM simply hopes that the local gradient holds true more globally.

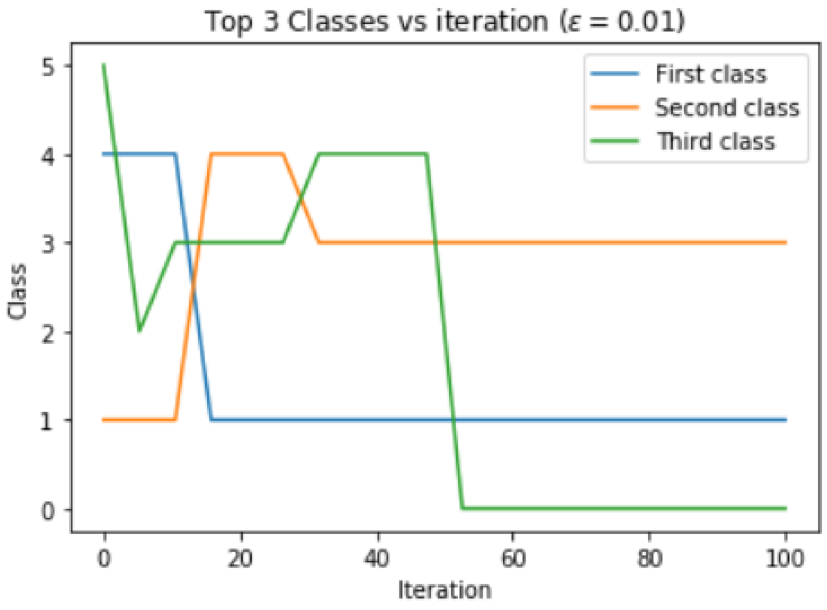


FIGURE 10. ResNet50's top three class predictions adversarial example varying with respect to varying iteration.

There are six different classes that appear either as the original classification or inside the top three class predictions for the adversarial model. They are as follows:

- 0: Terrapin
- 1: Leatherback Turtle
- 2: Hippopotamus
- 3: Loggerhead
- 4: Sea Lion
- 5: Otter

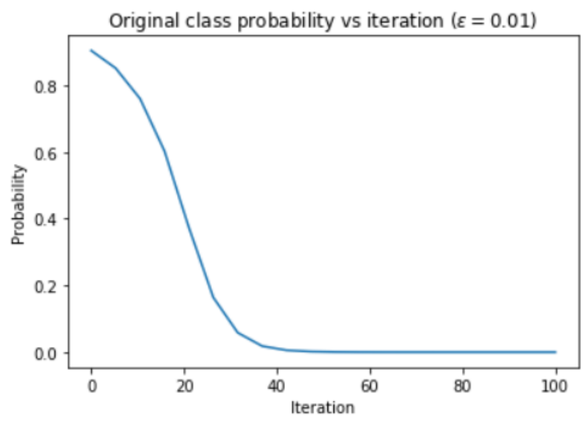
The sea lion again starts as the top class, but drops out by iteration 15, and is no longer part of the top three classes by iteration 50. Moreover, it can be seen that the confidence in the highest probability class continues to increase as iterations continue. Unlike the FGSM, local gradients are being updated so the direction of change is updated such that it is always pointing in the direction closest to the decision boundary. In the later epochs after the original class is no longer the most probable, this is clearly in the direction of the most probable class, so a monotonic increase can be observed there.

We now explore the results of some targetted IM attacks. We set the target to class 346 in ImageNet or the water buffalo.

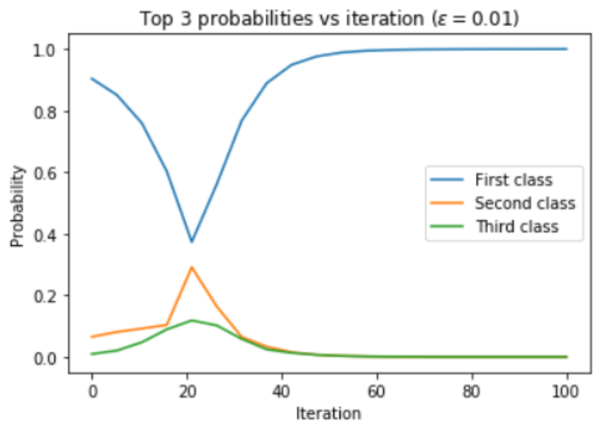
(A) Adversarial example generated from the original image using targetted IM on ResNet50 with 100 iterations on $\epsilon = 0.01$. This is classified as a water buffalo with confidence 0.999949.

(B) Pixel difference of the generated adversarial example in (A) with the original image. This is classified as a space shuttle with confidence 0.0675625.

FIGURE 11. Visual results of targetted FGSM attack on ResNet50 with 100 iterations on $\epsilon = 0.01$.



(A) ResNet50's prediction of the probability of the adversarial example being the originally classified class varying with respect to varying iteration.



(B) ResNet50's probabilities of the top three classes of the adversarial example varying with respect to varying iteration.

FIGURE 12. Variation of the confidence of ResNet50 in its top three predictions and of the original class as a function of iteration count.

While the rate of decline for the original class confidence is slower than with untargetted IM, the dominance of the confidence of the targetted class after a moderate number of iterations is much higher. Specifically, notice that the confidences for the second and third most probable classes converge before iteration 40 and are pushed very close to 0 quickly. Meanwhile the confidence for the target class dominates very near 1. Compared to the speed of convergence as well as the dominance of the confidence levels, targetted IM performs better that targetted FGSM in both cases.

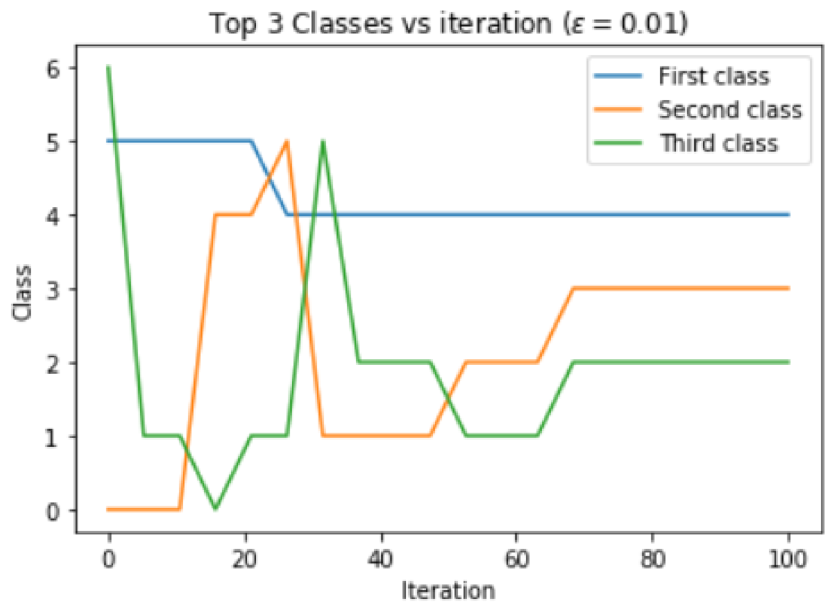


FIGURE 13. ResNet50's top three class predictions adversarial example varying with respect to varying iteration.

There are seven different classes that appear either as the original classification or inside the top three class predictions for the adversarial model. They are as follows:

- 0: Leatherback Turtle
- 1: Hippopotamus
- 2: Indian Elephant
- 3: Ox
- 4: Water Buffalo
- 5: Sea Lion
- 6: Otter

It can be observed that the change in top class is fast and dominant. Notably, the targetted class makes a sudden appearance in the top classes and overcomes the original class very quickly. Once that has occured, the second and third classes have much more shuffling than in targetted FGSM, but that is because the extraneous confidences are pushed so low that tiny perturbations on the scale of 10^{-6} can change a class's probability rank.

5. Conclusions and Discussion

It can be seen that both FGSM and IM have been implemented and used to create adversarial examples on ResNet50 trained on ImageNet classes in both the flavors of targetted attacks and untargetted attacks. While untargetted attacks performed more quickly in achieving an incorrect label, targetted attacks allowed for a selection of the class that the misclassification will trend towards. Similarly FGSM had the tradeoff of being faster than IM, but achieving results that fooled the neural network under a smaller confidence margin than IM did. However, all attack methods yielded good results and we have demonstrated an implementation that can be used for testing networks for robustness against adversarial example attacks.

There are some downfalls with the implementation. Specific to targetted attacks, not all target classes seem to work. More specifically, choosing certain classes that have negligible local gradient near the starting example leaves no way for the adversarial example to move towards the correct class. In cases like this, an adversarial example will not be able to be generated (it will actually just stay without perturbation since the gradient will be 0). This occurs when the implementation was tried on the least likely class at the starting point, but it is likely that it will also happen for other similarly improbable classes for the starting example.

Furthermore, the performance of the implementations and the selections of ϵ and iterations needed for good results will vary depending on the network as well as on the parameters chosen, including the starting example. This is simply due to different networks and starting seeds having completely different gradient landscapes. However, the concept is applicable to all networks, so it should simply be a matter of hyperparameter tuning to obtain good results similar to those demonstrated in the results section.

More improvements can be explored with the implementation of the algorithms. Currently the FGSM method suggests to use the signs of the gradients. However, this implies that the step updates will all be of size ϵ. This can be improved in two different ways. First, for IM, the step size can be improved and changed over iterations depending on the overall magnitude of the gradients. The magnitude information is being discarded and may harm convergence speed. Furthermore, it may also be possible to utilize the magnitudes of each entry of the gradient instead of the sign. Then entries that have higher magnitude command larger change than more negligible elements. This may again help convergence and these two methods may be areas for future exploration.

References

[1] I.J. Goodfellow, J. Shlens, C. Szegedy. "Explaining and Harnessing Adversarial Examples". arXiv:1412.6572. 2015.
[2] A. Kurakin, I.J. Goodfellow, S. Bengio. "Adversarial Machine Learning at Scale". arXiv:1611.01236. 2017
[3] N. Carlini, D. Wagner. "Towards Evaluating the Robustness of Neural Networks". arXiv:1608.04644. 2017.

Appendix

Listing 7. Python script for uploading local images to Google Colaboratory environment.

```
from google.colab import files

!rm my_img.jpg

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(name=fn,
      length=len(uploaded[fn])))
```

Listing 8. Python script for reversing ImageNet preprocessing of data to display images.

```
def plot_img(x):
    #Reverse preprocessing
    t = np.zeros_like(x[0])
    t[:,:,0] = x[0][:,:,2]
    t[:,:,1] = x[0][:,:,1]
    t[:,:,2] = x[0][:,:,0]
    plt.imshow(np.clip((t+[123.68, 116.779, 103.939]), 0, 255)/255)
    plt.grid('off')
    plt.axis('off')
    plt.show()
def show(x_adv):
  plot_img(x_adv)
  plot_img(x_adv - x)
  preds = model.predict(x_adv - x)
  print(e, 'Predicted:', decode_predictions(preds, top=3)[0])
```

Listing 9. Python script for reloading model for speed.

```
def reload_model():
  global model
  global K
  global sess
  time0 = time.time()
  K.clear_session()
  model = ResNet50(weights='imagenet')
  sess = K.get_session()
  time1 = time.time()
  print('reloading took: {} seconds'.format(time1 - time0))
```