

COMPUTING ADVERSARIAL PERTURBATIONS FOR SUCCESSFUL IMAGE CLASSIFIERS - A PYTHON IMPLEMENTATION OF DEEPFOOL -

Semester project submitted to

Signal Processing Laboratory (LTS4)
École Polytechnique Fédérale de Lausanne

Author: Eric Bezzam
Supervisor: Seyed Mohsen Moosavi-Dezfooli
Professor: Prof. Pascal Frossard

January 13, 2017

Contents

List of Figures	3
List of Tables	4
1 Introduction	5
2 Theory	6
2.1 DeepFool approaches	6
2.1.1 Multi-class	6
2.1.2 Binary	7
2.1.3 Multi-label	8
2.2 Gradient computation	8
2.2.1 Choice of δ	9
2.2.2 Random subspace approach	9
3 Implementation	11
3.1 General structure	11
3.2 Targeted classifiers	13
3.2.1 MNIST fully connected single layer	13
3.2.2 Keras (ResNet50, VGG16, VGG19)	13
3.2.3 Google's Inception v3	14
3.2.4 APIs (Clarifai, Google, IBM, Amazon, Microsoft)	15
3.3 Key considerations and observations	16
3.3.1 Input format	16
3.3.2 "Image friendly" values	16
3.3.3 Dealing with "disappearing" labels	17
3.3.4 Improving convergence	17
4 Results	19
4.1 Keras	19
4.1.1 ResNet50	19
4.1.2 VGG16	21
4.1.3 VGG19	22
4.2 Google's Inception v3	24
4.2.1 Multiclass	24
4.2.2 Binary	25
4.3 Clarifai API	27
4.3.1 Single label removal	27

CONTENTS

4.3.2 Multi-label removal	29
5 Conclusion	31
Bibliography	32

List of Figures

2.1	Delta versus MSE error.	9
4.1	Multiclass DeepFool applied to ResNet50 and pics_bmp/mushroom.bmp	20
4.2	Multiclass DeepFool applied to ResNet50 and pics_bmp/polar-bear-cropped.bmp	20
4.3	Multiclass DeepFool applied to ResNet50 and pics_bmp/soccer-ball.bmp	20
4.4	Multiclass DeepFool applied to VGG16 and pics_bmp/burrito_1.bmp	21
4.5	Multiclass DeepFool applied to VGG16 and pics_bmp/desk.bmp	21
4.6	Multiclass DeepFool applied to VGG16 and pics_bmp/television.bmp	21
4.7	Multiclass DeepFool applied to VGG19 and pics_bmp/burrito_1.bmp	22
4.8	Multiclass DeepFool applied to VGG19 and pics_bmp/mailbox.bmp	23
4.9	Multiclass DeepFool applied to VGG19 and pics_bmp/mashed-potato.bmp	23
4.10	Multiclass DeepFool applied to VGG19 and pics_bmp/minivan.bmp	23
4.11	Multiclass DeepFool applied to Inception v3 and pics_bmp/coffeepot.bmp	24
4.12	Multiclass DeepFool applied to Inception v3 and pics_bmp/forklift.bmp	25
4.13	Multiclass DeepFool applied to Inception v3 and pics_bmp/frying-pan.bmp	25
4.14	Binary DeepFool applied to Inception v3 and pics_bmp/backpack.bmp	26
4.15	Binary DeepFool applied to Inception v3 and pics_bmp/cauliflower_1.bmp	26
4.16	Binary DeepFool applied to Clarifai API and pics_bmp/backpack.bmp	28
4.17	Binary DeepFool applied to Clarifai API and pics_bmp/crane-building.pdf.bmp	28
4.18	Binary DeepFool applied to Clarifai API and pics_bmp/white-wolf-resize.bmp	28
4.19	Binary DeepFool applied to Clarifai API and pics_bmp/cucumber_1.bmp	29
4.20	DeepFool applied to the Clarifai API and cropped_panda.bmp	29

List of Tables

2.1	Summary of DeepFool algorithm variants.	6
2.2	Comparison of Image Recognition API's	10
4.1	ResNet50 results	19
4.2	VGG16 results	22
4.3	VGG16 results	22
4.4	Inception v3 Muliclass DeepFool results	24
4.5	Inception v3 Binary DeepFool results	26
4.6	Clarifai API results for <code>backpack.bmp</code>	27
4.7	Results for Clarifai API - Single Label Removal	27
4.8	Original results from Clarifai API for <code>cropped-panda.bmp</code>	29
4.9	DeepFool results for Clarifai API and <code>cropped-panda.bmp</code>	30

1 Introduction

Artificial intelligence is certainly a rising trend as computers have, in recent years, achieved near human performance when it comes to everyday tasks such as speech and image recognition. The release of the ImageNet database has thrust forward the area of image classification by providing a benchmark to compare the state-of-the-art classifiers [1]. Since 2010, the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) pits the best classifiers against each other. In 2012, the winning classifier with a top-five accuracy of 84.7 percent was by Krizhevsky, Sutskever, and Hinton [2]. They used a deep convolutional neural network (CNN), which has gone on to be used in many of the later submissions. The 2013 winner was Clarifai - now a company that has made their classifier into an API (Application Program Interface) product - achieving an accuracy of 88.7 percent [3][4]. The 2014 winner was Google's Inception (aka GooLeNet) classifier trumping the past winners with a top five accuracy of 93.33 percent [5]. It seems as though computers are already as good as humans when it comes to image recognition and object classification. Moreover, several of the labels in the ImageNet database would even be difficult for humans to discern, such as the difference between dog breeds.

However, there is more to the problem than meets the eye. Recent studies have shown that it is possible to obtain an *adversarial* perturbation, i.e. a noise tuned to a particular image and classifier, so that the perturbed, or modified image, is imperceptible from the original image *yet* the classifier yields a (sometimes completely) different label [6]. These studies raise concerns on the robustness of such classifiers. Can they be trusted if an imperceptible noise can completely throw off its results?

The goal of this semester project was to port the already implemented DeepFool algorithm [7] from Matlab to Python and to develop an implementation of a *blackbox* approach, i.e. a situation in which we do not have full access to the classifier structure. As well as applying this blackbox approach to “academic” networks, such as those submitted for the ILSVRC competition, we will also apply it to a commercial classifier - Clarifai - to gauge the robustness of products in the area of image classification and object recognition. An implementation of DeepFool for Google's Cloud Vision, IBM's Visual Recognition, Amazon Rekognition, and Microsoft's Computer Vision APIs is also made available but not extensively tested.

This report is divided as such: Chapter 2 will present the algorithmic details with regards to DeepFool and the blackbox approach; Chapter 3 will explain the implementation details; Chapter 4 presents results and observations from this project; finally, Chapter 5 summarizes the work from this semester project and discusses future work.

2 Theory

The full algorithmic details and mathematical explanation can be found in a paper by Moosavi-Dezfooli *et al* [6]. In this section, we will briefly describe the relevant theory, namely the variants of DeepFool depending on given information (glassbox vs. blackbox) and the desired goal (changing the top label, reducing the score of a label to a particular score, or reducing the score of multiple labels). By *glassbox*, we refer to a scenario in which we have the full structure of the classifier and its output labels while *blackbox* refers to a situation in which we only have access to the scores of a few output labels, as for an API like Clarifai [4], Google’s Cloud Vision [8], IBM’s Visual Recognition [9], Amazon Rekognition [10], and Microsoft’s Computer Vision [11].

Despite these slight differences, the general concept of DeepFool remains the same - compute an *adversarial perturbation* for a given classifier and image by greedily moving towards the boundaries of those labels different from the undesired one(s). For a given classifier, an adversarial perturbation is the *minimum* perturbation \mathbf{r} so that the estimated label $\hat{k}(\mathbf{x})$ for an image \mathbf{x} is changed [6]:

$$\mathbf{r} := \min \|\mathbf{r}\|_2 \text{ s.t. } \hat{k}(\mathbf{x} + \mathbf{r}) \neq \hat{k}(\mathbf{x}). \quad (2.1)$$

2.1 DeepFool approaches

Below is a summary of the different variants of DeepFool implemented and tested in this project.

	Short name	Task	Glassbox	Blackbox
Algorithm 1	Multi-class	Change top label	✓	
Algorithm 2	Binary	Reduce score of a particular label	✓	✓
Algorithm 3	Multi-label	Remove label(s) from top predictions		✓

Table 2.1 Summary of DeepFool algorithm variants.

With this mind, we will now present the algorithmic steps for each algorithm variant of DeepFool. The full proofs can be found in [6].

2.1.1 Multi-class

The steps of Algorithm 1 are taken from Algorithm 2 (DeepFool: multi-class case) of [6]. In line 4, the subscript of ∇f denotes the column of the Jacobian matrix. This approach can *only* be used in the glassbox scenario as it requires full knowledge of the labels and their corresponding scores. In the blackbox scenario, only the top N labels are returned, which may be problematic when computing the Jacobian, as is explained in Section 3.3.3.

```

input : vectorized image  $\mathbf{x}$ , classifier  $f$ .
output: perturbation  $\hat{\mathbf{r}}$ , perturbed image  $\mathbf{x}_p$ 
1 initialize  $\hat{\mathbf{r}} \leftarrow \mathbf{0}$ ,  $\mathbf{x}_p \leftarrow \mathbf{x}$ ;
2 while  $\hat{k}(\mathbf{x}_p) = \hat{k}(\mathbf{x})$  do
3   for  $k \neq \hat{k}(\mathbf{x})$  do
4      $\mathbf{w}'_k \leftarrow \nabla f_k(\mathbf{x}_p) - \nabla f_{\hat{k}(\mathbf{x})}(\mathbf{x}_p)$ ;
5      $f'_k \leftarrow f_k(\mathbf{x}_p) - f_{\hat{k}(\mathbf{x})}(\mathbf{x}_p)$ ;
6   end
7    $\hat{j} \leftarrow \arg \min_{k \neq \hat{k}(\mathbf{x})} \frac{|f'_k|}{\|\mathbf{w}'_k\|_2}$ ;
8    $\mathbf{r} \leftarrow \frac{|f'_j|}{\|\mathbf{w}'_j\|_2^2} \mathbf{w}'_j$ ;
9    $\mathbf{x}_p \leftarrow \mathbf{x}_p + \mathbf{r}$ ;
10   $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \mathbf{r}$ ;
11 end
return :  $\hat{\mathbf{r}}$ ,  $\mathbf{x}_p$ 

```

Algorithm 1: Multi-class DeepFool for changing the top label.

2.1.2 Binary

The algorithmic steps for applying DeepFool to reduce the score of a particular label can be found below. These are essentially the same steps as from Algorithm 1 (DeepFool for binary classifiers) of [6]. The main difference between this version of DeepFool and that of Algorithm 1 lies in the fact that we are treating the i^{th} output of a multi-class classifier as a binary classifier whose score we want to reduce below a certain threshold τ .

This algorithm can be used in the glassbox case when we would like to reduce the score of a particular label, which may not necessarily be the top label. This approach is particularly useful for the blackbox case as we may not have access to all the output labels and simply removing the targeted label by reducing the score until it is no longer in the top N predictions may be the best we can do in terms of “fooling” the classifier.

```

input : vectorized image  $\mathbf{x}$ , classifier  $f$ , label output index  $i$ , threshold  $\tau$ .
output: perturbation  $\hat{\mathbf{r}}$ , perturbed image  $\mathbf{x}_p$ 
1 initialize  $\hat{\mathbf{r}} \leftarrow \mathbf{0}$ ,  $\mathbf{x}_p \leftarrow \mathbf{x}$ ;
2 while  $f_i(\mathbf{x}_p) > \tau$  do
3    $\mathbf{r} \leftarrow \frac{f_i(\mathbf{x}_p)}{\|\nabla f_i(\mathbf{x}_p)\|_2^2} \nabla f_i(\mathbf{x}_p)$ ;
4    $\mathbf{x}_p \leftarrow \mathbf{x}_p + \mathbf{r}$ ;
5    $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \mathbf{r}$ ;
6 end
return :  $\hat{\mathbf{r}}$ ,  $\mathbf{x}_p$ 

```

Algorithm 2: Binary DeepFool for reducing the score of a particular label.

2.1.3 Multi-label

Another use case that may be of interest in the blackbox scenario is to remove multiple labels from the top N predictions. For instance, blackbox commercial classifiers tend to give multiple labels that are appropriate for a given image, such as “animal” and “panda” by Clarifai for the image in Figure 4.20 (see Table 4.8 for the full results). Moreover, we may be interested in removing multiple labels. The algorithmic steps for removing multiple labels from a classifier output can be found below where the classifier f returns a vector - \mathbf{F} - of the N top labels.

```

input : vectorized image  $\mathbf{x}$ , classifier  $f$ , labels to remove  $\mathbf{L}$ .
output: perturbation  $\hat{\mathbf{r}}$ , perturbed image  $\mathbf{x}_p$ 
1 initialize  $\hat{\mathbf{r}} \leftarrow \mathbf{0}$ ,  $\mathbf{x}_p \leftarrow \mathbf{x}$  ;
2 while  $\mathbf{L} \cap f(\mathbf{x}_p) \neq \emptyset$  do
3    $\mathbf{r} \leftarrow \mathbf{0}$ ;
4   for  $l \in \mathbf{L} \cap f(\mathbf{x}_p)$  do
5      $\mathbf{r} \leftarrow \mathbf{r} + \frac{f_l(\mathbf{x}_p)}{\|\nabla f_l(\mathbf{x}_p)\|_2^2} \nabla f_l(\mathbf{x}_p)$ 
6   end
7    $\mathbf{x}_p \leftarrow \mathbf{x}_p + \mathbf{r}$ ;
8    $\hat{\mathbf{r}} \leftarrow \hat{\mathbf{r}} + \mathbf{r}$ ;
9 end
return :  $\hat{\mathbf{r}}$ ,  $\mathbf{x}_p$ 

```

Algorithm 3: Multi-class DeepFool for removing multiple labels in blackbox scenario.

This algorithm could also be used in the glassbox scenario to reduce the score of multiple label or to remove multiple labels from the top N predictions.

2.2 Gradient computation

As we do not have full access to the classifier’s structure in the blackbox scenario, computing the gradient - as in done in line 4 of Algorithm 1, line 3 of Algorithm 2, and line 5 of Algorithm 3 - will have to be done numerically.

Finite different methods can be used to compute the gradients numerically. A common approximation for the derivative of a function f at point x is:

$$f'(x) \approx \frac{f(x + \delta) - f(x)}{\delta}. \quad (2.2)$$

However, the symmetric difference quotient below is generally a more accurate approximation than the above “one-sided” difference quotient:

$$f'(x) \approx \frac{f(x + \delta) - f(x - \delta)}{2\delta}. \quad (2.3)$$

This is due to the fact that the first-order errors cancel with the symmetric difference quotient approximation. In fact, TensorFlow’s function for computing the numerical gradient - `tf.test.compute_gradient` (which returns the symbolic gradient as well [12]) - uses the symmetric difference quotient approximation. This can be verified by running the script `test_numerical_gradient.py` in `mnist_softmax`.

Higher-order difference quotients could be used to reduce higher-order error terms; however, this quickly becomes impractical in our case as this would require more sample points. More sample points entails more *queries* from our classifier, but commercial classifiers have a *rate limit* that restricts the

number of times one may query the classifier. Therefore, we opt for the symmetric difference quotient in order to curb the (literal) cost of applying DeepFool to a particular image.

With our symmetric difference quotient in hand, we can go about computing the *Jacobian* matrix (as we have a vector input and a function that yields a vector output) of the first-order derivatives. The standard basis is used for computing the numerical Jacobian matrix. The standard basis for the vector space \mathbb{R}^N is given by the vectors \mathbf{e}_i for $i = 1, \dots, N$ where \mathbf{e}_i is equal to one at position i and zero elsewhere. The i^{th} row of the Jacobian matrix is then computed as such:

$$\nabla f_i(\mathbf{x}) \approx \frac{f(\mathbf{x} + \delta \cdot \mathbf{e}_i) - f(\mathbf{x} - \delta \cdot \mathbf{e}_i)}{2\delta}. \quad (2.4)$$

For an image of size (I, J, K) and C output labels, the Jacobian will be of size (N, C) , where $N = I \cdot J \cdot K$. Therefore, when using the symmetric difference quotient, we require $2N$ queries as each row of the Jacobian matrix requires two queries.

2.2.1 Choice of δ

The choice of δ has an influence on the error; for the symmetric difference quotient, the error is approximately proportional to δ^2 . Using Google’s Inception v3 network, we computed the deviation from the symbolic (ground truth) gradient values. Figure 2.1 is a result of running the following script (in `imagenet_inception`):

```
» python plot_numerical_gradient_error.py
```

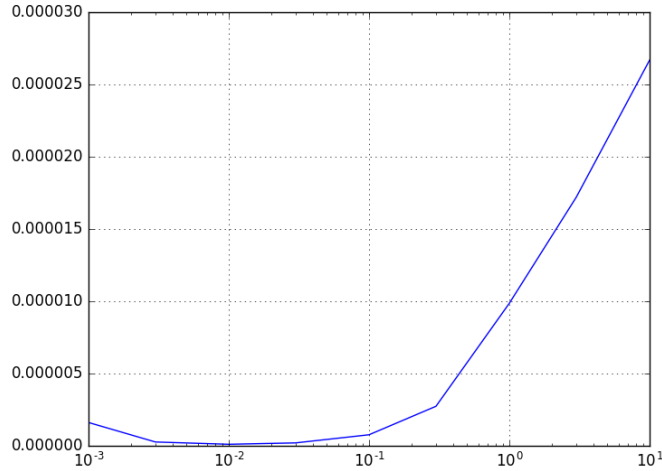


Figure 2.1 Delta versus MSE error between symbolic and numerical gradient of Inception v3.

2.2.2 Random subspace approach

As mentioned earlier, commercial classifiers have a rate limit, restricting the number of queries that can be performed. If we consider a rather small RGB image with dimensions $(224, 224, 3)$ (a typical input size for ImageNet classifiers), we would require $2 \cdot 244 \cdot 244 \cdot 3 = 301056$ entries per iteration of DeepFool. Even if we did not have a constraint on the number of allowed queries, this would take a very long time to compute. Table 2.2 summarizes the rate limit constraints of different APIs.

Fortunately, there is a way to reduce this computational and monetary cost in running DeepFool. Instead of using the standard basis of dimension N , we can estimate the numerical Jacobian with a

	# free queries	Additional queries	Notes
Clarifai	5000 per month	\$1.20 per 1000	
Google's Cloud Vision	1000 per month	\$1.50 per 1000	Need to open billing account even for free use.
IBM's Visual Recognition	250 per day	\$0.002 per Image	Does not support BMP and TIFF.
Amazon Rekognition	5000 per month for first 12 months	\$1.00 per 1000	Need to open billing account even for free use. Does not support BMP and TIFF.
Microsoft's Computer Vision	5000 per month, 20 per minute	\$1.50 per 1000	
CloudSight	500 for one-time free trial	\$49 for 800; \$149 for 3000; \$399 for 10000; \$1499 for 50000	Takes 6-12 seconds to receive a completed response. Does not give labels with a corresponding score but rather a textual description.

Table 2.2 Comparison of Image Recognition API's. An interesting comparison can be found at this blog post [13].

random subspace of dimension $D \ll N$. However, this comes at the cost of increasing the perturbation norm by a factor of $\sqrt{N/D}$ [14].

The random subspace - \mathbf{P}_S - is computed from a random matrix of size (N, D) with the entries drawn from normal distribution of mean 0 and variance 1. It is then made into an orthonormal basis, e.g. by means of the Gram-Schmidt process. The Jacobian matrix in this reduced subspace will now be of size (D, C) and each row will be computed as such:

$$\nabla f_{S,i}(\mathbf{x}) \approx \frac{f(\mathbf{x} + \delta \cdot \mathbf{s}_i) - f(\mathbf{x} - \delta \cdot \mathbf{s}_i)}{2\delta}. \quad (2.5)$$

The approximated full numerical Jacobian \mathbf{J} can then be computed from the Jacobian matrix in the random subspace \mathbf{J}_S with a simple projection:

$$\mathbf{J} = \mathbf{P}_S \times \mathbf{J}_S \quad (2.6)$$

Our objective from Equation 2.1 is subsequently modified to the following expression as our perturbation will be computed from the vectors spanned by the random subspace \mathcal{S} :

$$\mathbf{r}_S := \min_{\mathbf{r} \in \mathcal{S}} \|\mathbf{r}\|_2 \text{ s.t. } \hat{k}(\mathbf{x} + \mathbf{r}) \neq \hat{k}(\mathbf{x}) \quad (2.7)$$

3 Implementation

Python was chosen as the development language for the following reasons:

1. Easy to use for prototyping and allows for “readable” code;
2. Widely used in the open source and developer community, which allows us to make use of state-of-the-art libraries such as Numpy [15], TensorFlow [16], Theano [17], and Keras [18];
3. Image Classification APIs are available in Python.

In this section, we will describe the overall structure of the code, which should also give insight into how it can be extended for applying DeepFool to other classifiers.

3.1 General structure

At the center of everything is the file `deepfool.py` which contains an abstract class entitled `DeepFool`. There are two methods for applying DeepFool in the glassbox and blackbox scenarios: `_apply_glassbox` and `_apply_blackbox` respectively. Running `_apply_glassbox` will guide the user through an interface which prints (at most) 30 labels with the highest prediction scores. The user will be prompted to select one of two tasks:

1. Change the top label;
2. Reduce the score of a particular label.

Selecting ‘Change the top label’ will apply the Algorithm 1 variant of DeepFool. Choosing ‘Reduce the score of a particular label’ will prompt the user to select a label to reduce and a corresponding threshold. Consequently, the Algorithm 2 variant of DeepFool will be applied.

Running `_apply_blackbox` will similarly guide the user through an interface which prints (at most) 30 labels with the highest prediction scores. The user will be prompted to select one of two tasks:

1. Reduce the score of a particular label;
2. Remove label(s) from the top predictions.

Reducing the score of a particular label will prompt the user to select a label to reduce and a threshold. The Algorithm 2 variant of DeepFool will then be applied. Removing label(s) from the top predictions will prompt the user to select the desired label(s) to remove. Consequently the Algorithm 3 variant of DeepFool will be applied.

The following methods for debugging have also been implemented:

- `compare_with_original`: compare the (at most) 30 top predictions labels of a given image (default is the perturbed image) with those of the original image.
- `compare_with_random_noise`: compute multiple (default 10) perturbed images by adding random noise that has the same norm as that of the adversarial noise; then compute the predictions of these “randomly” perturbed images to ensure that the score reduction / label(s) removal performed by

the adversarial perturbation cannot be done by just any randomly generated noise.

The rest of the methods in `deepfool.py` serve for the implementation of the command line user interface (i.e. printing prompts and results nicely for the user) and the variants of DeepFool (e.g. computing the Jacobian and updating variables for each iteration).

For each classifier, a child to `DeepFool` has been created. The unique aspects of each classifier’s implementation will be described in the following sections. Key differences include the structure of the input to the classifier, the structure of the classifier’s output, how the classifier’s predictions are obtained, and how to parse the results. As each classifier returns scores within $[0, 1]$ (i.e. the output of a softmax layer for the academic networks or confidence scores for the APIs), we apply a natural log to the output in order to approximately invert this output layer as the DeepFool algorithm does not perform well on such outputs.

Each classifier’s DeepFool implementation requires a method called `calculate_predictions`, which returns the prediction results in the form of a dictionary, with the label being the *key* and the score being the *value*. This is the only method required by the functions in `deepfool.py` in order to apply the appropriate DeepFool variant. Moreover, `calculate_predictions` must have four inputs:

1. `input_image`: vectorized image whose predictions will be calculated; if it needs to be reshaped or saved as an image file in order to query the classifier, this should be done within `calculate_predictions`.
2. `num_pred`: number of labels to be returned by the classifier; this is more useful for the academic networks if one wishes to just know the top N labels; does not need to be used by `calculate_predictions`.
3. `print_results`: Boolean variable which can be used to call the DeepFool method `_print_prediction_results` in order to print the results nicely for the user.
4. `image_friendly`: Boolean which can be used to notify `calculate_predictions` whether or not the image values should be quantized into 256 values before calculating the predictions; this is not needed for the APIs as the image values have to be anyway saved as an image to query the classifier which will ultimately lead to quantization; in the glassbox network, however, it helps avoid this quantization; more on this in Section 3.3.1.

We have also implemented additional functions for convenient use and debugging such as:

- `set_input_image`: give an image path to set the input image; this removes the need of having to create a separate DeepFool object for each image which can be memory intensive due to the size of the classifiers (when using the academic networks).
- `visualize_image`: plot the image using the `matplotlib.pyplot.imshow` function; needs to be done differently for each image due to the differences in the value ranges and dimension ordering (as the values passed to `matplotlib.pyplot.imshow` must be of type `float`, within the range $[0, 1]$, and have the RGB channels as the third dimension).
- `save_image`: save a given image (default perturbed image) to disk with the `scipy.misc.imsave` function; needs to be done differently for each image due to the differences in the value ranges and dimension ordering (as the values passed to `scipy.misc.imsave` must be of type `uint8`, within the range $[0, 255]$, and have the RGB channels as the third dimension).

3.2 Targeted classifiers

3.2.1 MNIST fully connected single layer

In `mnist_single_layer`, there is (as the name suggests) an MNIST neural network classifier with a single layer, with a softmax at the output. It was implemented in TensorFlow, by following one of the tutorials [19]. Working with this small network was useful when testing the very first implementations of DeepFool and ensuring that the gradient values were computed correctly, as the symbolic gradients can be computed quickly. In order to apply DeepFool to this classifier, one simply needs to run the script `fool_mnist_single_layer.py`. First DeepFool with the symbolic Jacobian matrix will be applied, then the numerical Jacobian, and finally the subspace method with $D = 200$. A different image can be used by setting the command line argument `-i` to a value within $[0, 9999]$.

3.2.2 Keras (ResNet50, VGG16, VGG19)

The `keras` library allows us to conveniently query three classifiers - ResNet50, VGG16, and VGG19 - trained on the ImageNet database of 1000 output labels [18][20]. We decided to use Theano as a backend when running Keras.

The pixel values must be within the range of $[0, 255]$, and the input dimensions for querying the classifiers are of the shape: $(N, 3, 224, 224)$, where N is the number of images. Multiple images can be sent to a classifier (so that predictions can be computed in parallel). We will, however, just be computing the predictions for one image at a time. This feature could be taken advantage of to speed up the computation of the Jacobian matrix.

With the pre-trained models by Keras [20], it is possible to emulate the blackbox scenario by setting the number of predictions parameters - `num_pred` - in the constructors to a value below 1000. At the moment, the Jacobian for the Keras model can only be computed numerically (full or subspace method) so the only difference between the glassbox and blackbox scenarios lies in the knowledge of the full output label set and `image_friendly` always being set to `True` for the blackbox scenario.

To apply DeepFool on one of the Keras models, one simply needs to go to the `imagenet_keras` directory and run the following script:

```
» python fool_imagenet_keras.py -f <image_path>
```

where `<image_path>` is the path to the desired file, preferably in an uncompressed format such as BMP to avoid artifacts from lossy compression. The user will then be prompted which classifier they would like to fool and in what manner.

By default for each classifier (and for the results in Section 4.1), the subspace dimension is set to $D = 100$, the overshoot parameter (described in Section 3.3.4) is set to 1, and the symmetric difference quotient parameter is set to $\delta = 0.01$. It is possible to set these values and other parameters as command line arguments with the above script:

1. Image file path: `-f` or `--image_path=`,
2. Model: `-m` or `--model=`; possible choices are `resnet50`, `vgg16`, and `vgg19`,
3. Subspace dimension D : `-s` or `--sub_dim=`,
4. Delta δ : `-d` or `--delta=`,
5. Overshoot: `-o` or `--overshoot=`,
6. Maximum number of iterations: `-i` or `--max_iter=`,
7. Number of predictions returned by the classifier: `-p` or `--num_pred=`; this can be used to “simulate”

the blackbox case where we do not have the full set of output labels.

3.2.3 Google’s Inception v3

Instead of using the Inception network via Keras, we used the trained network directly provided through TensorFlow’s documentation [5]. This was mainly done as the gradients can be more conveniently accessed and computed by working with Inception v3 directly with TensorFlow.

Using the network “out of the box” was not necessarily straightforward as the example in the given `classify_image.py` file only shows how to provide an input that is a path to a JPG file. Moreover, saving to JPG in order to query the classifier would hamper the DeepFool process as JPG is a lossy compression, thus introducing artifacts. Therefore, we need go past this *operation* in the TensorFlow graph of the Inception model and find the actual entry point into the classifier. Unfortunately, there is not much documentation on each Tensor and Operation in the Inception graph. However, with the IPython shell we can investigate this on our own by first loading the model with the `create_graph` function inside `classify_image.py`:

```
» sess = tf.InteractiveSession()
» create_graph()
```

and by running:

```
» [n.name for n in tf.get_default_graph().as_graph_def().node]
```

The last command prints all the Tensors in the Graph to the Terminal output. After some trial and error and checking the shapes of the Tensors, we concluded that the input to the neural network is at the Tensor `Mul:0`. It has the shape $(1, 299, 299, 3)$, and the input pixel values must be within the range $[-1, 1]$.

The output of the classifier can be taken at the Tensor `softmax:0` which then needs to be (approximately) inverted with the natural log by adding a TensorFlow operation as such:

```
» output_tensor = sess.graph.get_tensor_by_name('softmax:0')
» classifier = tf.log(output_tensor)
```

One could also obtain the output before the softmax layer with the following commands:

```
» weights = sess.graph.get_tensor_by_name('softmax/weights:0')
» biases = sess.graph.get_tensor_by_name('softmax/biases:0')
» x = sess.graph.get_tensor_by_name('pool_3/_reshape:0')
» classifier = tf.matmul(x, weights) + biases
```

To obtain the predictions of an image, it must first be resized to the shape $(1, 299, 299, 3)$ and fed to the `classifier` Tensor as such (where `image_data` is the reshaped data within the range $[-1, 1]$):

```
» pred = classifier.eval(feed_dict={'Mul:0':image_data})
```

To ensure that the Inception v3 model is properly downloaded to one’s computer, the script `label.py` in `imagenet_inception` can be run:

```
» python label.py <jpg_file_path>
```

After downloading the Inception v3 model and placing it in `imagenet_inception`, DeepFool can be applied to an image by running the following script:

```
» python fool_imagenet_inception.py -f <image_path>
```

where `<image_path>` is the path to the desired file.

By default (and for the results in Section 4.2), the subspace dimension is set to $D = 100$, the overshoot parameter to 0.005, and the symmetric difference quotient parameter is set to $\delta = 0.01$. It is possible to set these values and other parameters as command line arguments with the above command:

1. Image file path: `-f` or `--image_path=`,
2. Subspace dimension D : `-s` or `--sub_dim=`,
3. Delta δ : `-d` or `--delta=`,
4. Overshoot: `-o` or `--overshoot=`,
5. Maximum number of iterations: `-i` or `--max_iter=`,
6. Number of predictions returned by the classifier: `-p` or `--num_pred=`; this can be used to “simulate” the blackbox case where we do not have the full set of output labels.

3.2.4 APIs (Clarifai, Google, IBM, Amazon, Microsoft)

As all the APIs represent a blackbox scenario, we do not know how the input is processed and fed to the classifier. Therefore, as an input shape we simply use that of our image we are applying DeepFool to. For this reason, it may be desired to crop the targeted object to fool and perhaps subsample the image to reduce the number of input dimensions.

Each time that we would like to query the classifier, e.g. when computing the Jacobian matrix, we must save the perturbed image as a valid uncompressed image format, e.g. BMP, TIFF, or PNG. Moreover, the application of DeepFool to each API’s classifier only differs in three aspects:

1. Authentication: sending requests to an API typically requires some sort of authentication with an “API Key”.
2. The functions / methods used for querying the classifier.
3. Parsing the response returned by the API.

As these are the only differences, we have created an `api` class (which is a child of `DeepFool`) from which each API’s DeepFool implementation inherits from. Below is the name of the file and class for each API’s DeepFool implementation (all in the folder `deepfool`).

- Clarifai: `clarifai_api`
- Google: `cloud_vision`
- IBM: `visual_recognition`
- Amazon: `amazon_rekognition`
- Microsoft: `microsoft_cv`

Upon inspection of the each class’ implementation, one may observe that there are only two functions: the constructor for setting up the authentication and `_query_api` for querying the API, parsing the response, and applying log to “invert” the softmax. In reality, the outputs of these APIs are not really from a softmax layer as the scores do not sum to one, but the natural log helps to emphasize the difference between the scores to aid DeepFool.

In each API's folder (same name as the classes above), there is a `README.md` file that explains how to set up the authentication. Moreover, there is a `label.py` file to obtain the API's response on a desired file as such:

```
» python label.py <image_path>
```

We only gathered results from the Clarifai API as they were the most generous in offering free queries. In fact, they did not charge from the free query limit until December as they were operating under a 'developer preview' mode. Nevertheless, the other APIs are also ready for applying DeepFool.

By default (and for the results in Section 4.3), we use a higher $\delta (= 0.5)$ for the symmetric difference quotient to ensure that the deviated image in the difference quotient results in a change when the pixels are quantized (into 256 values) and saved as an image. When using $\delta = 0.01$ as for the previous classifiers, the numerical gradient was always zero because this value of δ was not high enough to yield a different image. The default subspace dimension is set to $D = 25$ as to not use too many credits (as a reminder the Clarifai API allows 5000 free queries and each iteration of DeepFool requires $2D + 1$ queries).

To apply DeepFool on the Clarifai API (or another API), one simply needs to go to the corresponding directory and run:

```
» %run fool_<api_name>.py -f <image_path>
```

where `<image_path>` is the path to the desired file. Please refer to Table 2.2 to see which file formats are accepted by each API. It is possible to set the following parameters as command line arguments with the above command:

1. Image file path: `-f` or `--image_path=`,
2. Subspace dimension D : `-s` or `--sub_dim=`,
3. Delta δ : `-d` or `--delta=`,
4. Overshoot: `-o` or `--overshoot=`,
5. Maximum number of iterations: `-i` or `--max_iter=`

3.3 Key considerations and observations

3.3.1 Input format

Different classifiers require different formats for their inputs: image shape of $(1, 3, 224, 224)$ with values within $[0, 255]$ for the Keras models; image shape of $(1, 299, 299, 3)$ with values within $[-1, 1]$ for the Inception v3 model; and an image file (of a particular format) for the APIs. However, for DeepFool we require a vectorized version of the image.

To accommodate such differences, we implemented a method for each classifier called `_prepare_input` that reshapes the vectorized image and for the APIs saves it as a file. Moreover, for the methods mentioned earlier (`set_input_image`, `visualize_image`, and `save_image`), we had to accommodate such differences by implementing those functions in the child class rather than the parent class.

3.3.2 "Image friendly" values

Ultimately, each pixel value will be represented by 8 bits, which means it can take on $2^8 = 256$ different values, namely all integers within the range $[0, 255]$. However, DeepFool works with float values. There-

fore, when querying a classifier for the prediction results of a *realistic* image, it is necessary that the pixel values only take on integer values between $[0, 255]$.

This is no concern for the APIs as saving the image to a valid image format will already ensure this. However, when working with the classifier directly, as with the Keras models and Inception v3, they will accept any float value as input. When computing the Jacobian to find the adversarial perturbation, such values could be used to get better adversarial perturbation (although not in-line with what would be allowed in the blackbox case). However, when computing the prediction scores of the perturbed image (such as at the end of a DeepFool iteration), it is important to have “image friendly” values to reflect what the scores would be for an actual image.

To accommodate for this, we have added a parameter named `image_friendly` to the `_prepare_input` methods of the Keras and Inception models. For Keras, this implies rounding the values as they are already within the range of $[0, 255]$ and casting them as `uint8`. For the Inception model, this means shifting the image values from the range $[-1, 1]$ to $[0, 255]$ and then casting them as `uint8`.

3.3.3 Dealing with “disappearing” labels

When working in the blackbox scenario, we only have access to a certain number of labels, e.g. 20 for the Clarifai API. Some of these labels, however, may come and go when we add perturbations, e.g. when computing the numerical Jacobian matrix.

We deal with these “disappearing” labels in a very simple manner. Suppose our blackbox classifier outputs only the top C labels; then we will attempt to construct a Jacobian matrix of size (D, C) where D is either the length of the vectorized image or the subspace dimension. We set the C classes as those labels acquired when querying the classifier before the computation of the Jacobian. If during the Jacobian computation, a particular label is not among the top C labels anymore when querying the classifier to compute the symmetric difference quotient, then we simply set the value to the minimum score of the C output labels. As the label has been removed it cannot be larger than this minimum value. As there is no other knowledge we have about the new score of this particular label, this choice serves as an appropriate estimate.

If our goal is to remove a particular label (or labels) and we happen to accomplish this during the Jacobian computation with a particular subspace vector, we may wish to prematurely terminate the DeepFool process. If this is the case, we can set `quit_early = True` when running `apply`.

3.3.4 Improving convergence

overshoot parameter

In order to reduce the number of iterations so that either the top label changes or the score of a particular label reduces under a certain threshold, we can use an `overshoot` factor (a parameter for the `apply` method). This factor will multiply the perturbation computed at the current iteration (line 8 of Algorithm 1, line 3 of Algorithm 2, before line 7 of Algorithm 3) by $(1 + \text{overshoot})$. At the moment, this `overshoot` parameter is chosen by trial and error. Ideally, it should be very small because the larger it is, the larger the perturbation norm will be, causing the added perturbation to be (possibly) more noticeable.

It is best to start with a small `overshoot` (e.g. 1 for the Keras models since the range of values is within $[0, 255]$ and 0.005 for Inception since the range of values is within $[-1, 1]$). If the score is not reducing at the desired rate, one can stop the process and continue the DeepFool process with a larger `overshoot` factor by setting the `cont` parameter to `True`, e.g. as such (where `df` is a DeepFool object):

```
» df.apply(overshoot=0.1, cont=True)
```

For future improvements to the implementation, it would be ideal to have a method for determining the appropriate `overshoot` value.

Changing subspace vectors

Another observed “trick” that could help when the reduction of the score starts to slow down is to change the random subspace vectors. This can be done with the `set_subspace_dimension` method, e.g. as such (where `df` is a `DeepFool` object):

```
» df.set_subspace_dimension(sub_dim=100)
```

Interrupting a `DeepFool` process to change the subspace vectors has not been done with the results in the following section.

4 Results

A small test set was created by downloading several images from the web by doing a Google Images search of some of the labels in the ImageNet database. These images can be found in the folder `pics`. These images can be converted to a desired format with the `convert_images.py` script by running the following command in the Terminal:

```
» python convert_images.py -e <EXTENSION>
```

The converted files are then stored in the folder `pics_<EXTENSION>`.

The IPython Shell was used for testing and obtaining the results the below. It is possible to also use the regular Python shell by replacing `%run` with `python`. Either shell makes the testing and debugging convenient as variables are stored in the workspace. All the images below are displayed in their original size.

4.1 Keras

DeepFool was applied for each Keras model on 3-4 images. In the following subsections, we will show the results for these images and comment on how the labels were changed due to the perturbation. The full results can be found in `report_results/imagenet_keras`. One important point is that DeepFool is applied to a *resized* version of the original image from `pics_bmp` as the input the classifiers must have a shape of (3,224,224). These resized images are stored in `report_results/imagenet_keras` with a prefix of `ORIG_`.

4.1.1 ResNet50

Multiclass DeepFool (Algorithm 1) was applied to the following images (in `pics_bmp`) and ResNet50.

Image	mushroom.bmp	polar-bear-cropped.bmp	soccer-ball.bmp
Original top label	plate	ice_bear	soccer_ball
Original score	0.33189	0.87409	0.70333
Perturbed score	0.26001	0.26068	0.35640
Perturbed top label	cauliflower	miniature_poodle	football_helmet
Original score	0.17047	0.02112	0.10522
Perturbed score	0.26234	0.26114	0.35669
Image norm	64768	69388	40158
Perturbation norm	758	1546	1464

Table 4.1 ResNet50 results for subspace dimension $D = 100$ and overshoot = 1.

For each of the images in Table 4.1, DeepFool was able to modify the original top label. We can observe from the corresponding images below that the added noise is practically imperceptible. Moreover, adding random noise of the same norm as that of the adversarial perturbation (by running the method `compare_with_random_noise`) was not able to change the top label.

The full results can be found in `report_results/imagenet_keras/resnet50/<Image>_results`.



(a) Original, top label of 'plate'



(b) Perturbed, top label of 'cauliflower'

Figure 4.1 Multiclass DeepFool applied to ResNet50 and `pics_bmp/mushroom.bmp`



(a) Original, top label of 'ice_bear'



(b) Perturbed, top label of 'miniature_poodle'

Figure 4.2 Multiclass DeepFool applied to ResNet50 and `pics_bmp/polar-bear-cropped.bmp`



(a) Original, top label of 'soccer_ball'



(b) Perturbed, top label of 'football_helmet'

Figure 4.3 Multiclass DeepFool applied to ResNet50 and `pics_bmp/soccer-ball.bmp`

4.1.2 VGG16

Multiclass DeepFool (Algorithm 1) was applied to the following images (in `pics_bmp`) and VGG16.



(a) Original, top label of 'burrito'



(b) Perturbed, top label of 'chambered_nautilus'

Figure 4.4 Multiclass DeepFool applied to VGG16 and `pics_bmp/burrito_1.bmp`



(a) Original, top label of 'desk'



(b) Perturbed, top label of 'barbershop'

Figure 4.5 Multiclass DeepFool applied to VGG16 and `pics_bmp/desk.bmp`



(a) Original, top label of 'monitor'



(b) Perturbed, top label of 'library'

Figure 4.6 Multiclass DeepFool applied to VGG16 and `pics_bmp/television.bmp`

For each of the images, DeepFool was able to modify the original top label. Just as for the ResNet50 perturbed images, we can observe that the added noise is practically imperceptible. Moreover, adding

Image	burrito_1.bmp	desk.bmp	television.bmp
Original top label	burrito	desk	monitor
Original score	0.15171	0.33112	0.54012
Perturbed score	0.12003	0.18990	0.33939
Perturbed top label	chambered_nautilus	barbershop	library
Original score	0.09869	0.09479	0.13350
Perturbed score	0.12823	0.20496	0.34182
Image norm	47001	49445	40158
Perturbation norm	334	1163	1464

Table 4.2 VGG16 results for subspace dimension $D = 100$ and overshoot = 1.

random noise of the same norm as that of the adversarial perturbation was not able to change the top label. Table 4.2 summarizes the results for the above images. The full results can be found in `report_results/imagenet_keras/vgg16/<Image>_results`.

4.1.3 VGG19

Multiclass DeepFool (Algorithm 1) was applied to the following images (in `pics_bmp`) and VGG19.

Image	burrito_1.bmp	mailbox.bmp	mashed-potato.bmp	minivan.bmp
Original top label	burrito	mailbox	spaghetti_squash	minivan
Original score	0.28946	0.26057	0.51727	0.38923
Perturbed score	0.12560	0.12700	0.38240	0.20874
Perturbed top label	pomegranate	swing	toilet_seat	racer
Original score	0.08046	0.03987	0.29055	0.09963
Perturbed score	0.12576	0.14071	0.40227	0.20942
Image norm	47001	49745	64443	40938
Perturbation norm	1276	1870	567	1677

Table 4.3 VGG19 results for subspace dimension $D = 100$ and overshoot = 1.



(a) Original, top label of 'burrito'



(b) Perturbed, top label of 'pomegranate'

Figure 4.7 Multiclass DeepFool applied to VGG19 and `pics_bmp/burrito_1.bmp`

For each image, DeepFool was able to modify the original top label. Although the original top label



(a) Original, top label of 'mailbox'



(b) Perturbed, top label of 'swing'

Figure 4.8 Multiclass DeepFool applied to VGG19 and `pics_bmp/mailbox.bmp`

(a) Original, top label of 'spaguetti_squash'



(b) Perturbed, top label of 'toilet_seat'

Figure 4.9 Multiclass DeepFool applied to VGG19 and `pics_bmp/mashed-potato.bmp`

(a) Original, top label of 'minivan'



(b) Perturbed, top label of 'racer'

Figure 4.10 Multiclass DeepFool applied to VGG19 and `pics_bmp/minivan.bmp`

for `mashed-potato.bmp` is incorrect, the inside of a spaghetti squash does have a similar appearance to mashed potato. Just as for the ResNet50 and VGG16 perturbed images, we can observe from the corresponding images below that the added noise is practically imperceptible. Likewise, adding random noise of the same norm as that of the adversarial perturbation was not able to change the top label. Table 4.3 summarizes the results for the above images. The full results can be found in `report_results/imagenet_keras/vgg16/<Image>_results`.

4.2 Google’s Inception v3

Multiclass DeepFool was applied on 3 images and Binary DeepFool (for reducing a particular label’s score) on 2 images. In the following subsections, we will show the results for these images and comment on how the labels were changed due to the perturbation. The full results can be found in `report_results/imagenet_inception`. One important point is that DeepFool is applied to a *resized* version of the original image from `pics_bmp` as the input to the Inception classifier must have a shape of $(299, 299, 3)$. These resized images are stored in `report_results/imagenet_inception` with a prefix of `ORIG_`.

4.2.1 Multiclass

Multiclass DeepFool (Algorithm 1) was applied to the following images (in `pics_bmp`) and the Inception v3 classifier.

Image	coffeepot.bmp	forklift.bmp	frying-pan.bmp
Original top label	coffeepot	forklift	frying-pan
Original score	0.80178	0.80167	0.61649
Perturbed score	0.37249	0.20787	0.33429
Perturbed top label	roisserie	golfcart, golf cart	measuring cup
Original score	0.02097	0.01835	0.05454
Perturbed score	0.37568	0.20893	0.33756
Image norm	333.13254	290.88104	280.80172
Perturbation norm	14.81096	44.84579	7.39622

Table 4.4 Inception v3 Multiclass DeepFool results for subspace dimension $D = 100$ and overshoot = 0.005.

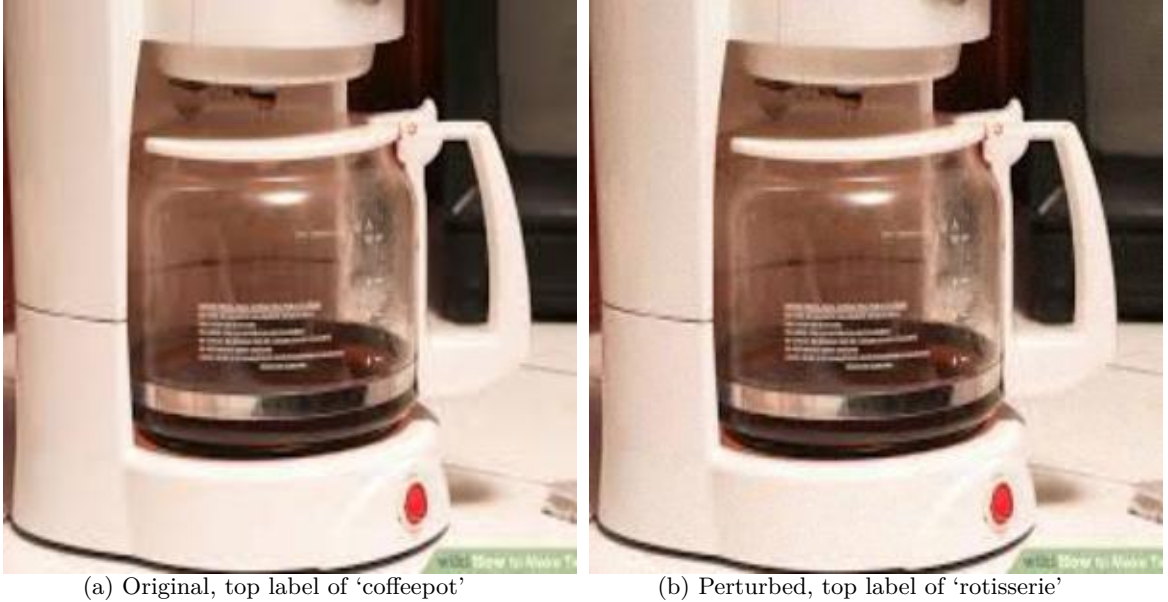


Figure 4.11 Multiclass DeepFool applied to Inception v3 and `pics_bmp/coffeepot.bmp`



Figure 4.12 Multiclass DeepFool applied to Inception v3 and `pics_bmp/forklift.bmp`

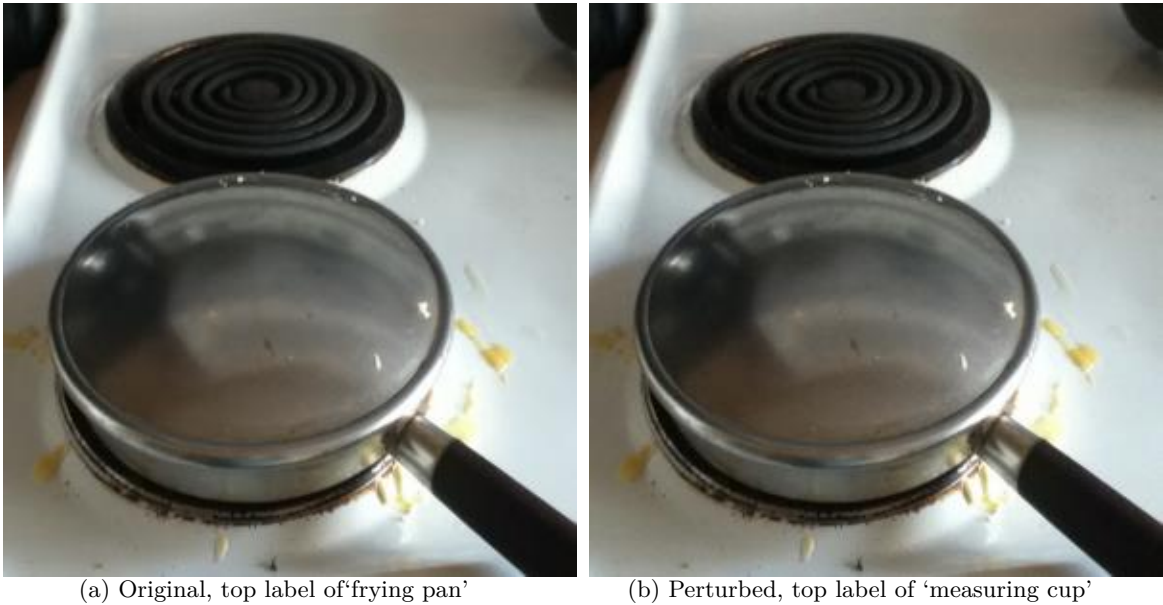


Figure 4.13 Multiclass DeepFool applied to Inception v3 and `pics_bmp/frying-pan.bmp`

For each of the images, DeepFool was able to modify the original top label. We can observe from the corresponding images that the added noise is quite small; for `forklift.bmp` the added noise starts to be noticeable. Moreover, adding random noise of the same norm as that of the adversarial perturbation was not able to change the top label. Table 4.4 summarizes the results for the above images. The full results can be found in `report_results/imagenet_inception/<Image>_results`.

4.2.2 Binary

With the Inception v3 classifier, we also test the Binary DeepFool variant. As we will see later on with the Clarifai classifier, sometimes the label humans might associate with an image may not be the top label. Therefore, we may wish to reduce a label that is not the top one; in this case, we would need to apply Algorithm 2. For the images below, the “true” label was not the top label. Nonetheless, we would like to reduce the score of the true label to further worsen the classifier’s results.

4 RESULTS

Image	backpack.bmp	cauliflower_1.bmp
Top label	bulletproof vest	coral fungus
Original score	0.46256	0.49046
Perturbed Score	0.94858	0.86562
Targeted label	backpack	cauliflower
Original score	0.1508	0.28117
Perturbed Score	0.00519	0.00107
Image norm	399.87729	324.13108
Perturbation norm	28.97545	43.90304

Table 4.5 Inception v3 Binary DeepFool results for subspace dimension $D = 100$ and overshoot = 0.005.



(a) Original, 0.1508 as 'backpack'



(b) Perturbed, 0.00519 as 'backpack'

Figure 4.14 Binary DeepFool applied to Inception v3 and pics_bmp/backpack.bmp



(a) Original, 0.28117 as 'cauliflower'



(b) Perturbed, 0.00107 as 'cauliflower'

Figure 4.15 Binary DeepFool applied to Inception v3 and pics_bmp/cauliflower_1.bmp

4.3 Clarifai API

4.3.1 Single label removal

The Algorithm 2 DeepFool variant was applied on 4 images to reduce the scores of a particular label. Compared with the previous classifiers, Clarifai tends to give more general terms as labels. For the example, the results for the original image in Figure 4.16 can be seen in Table 4.6. Our goal will be to remove a particular with Algorithm 2 by setting the threshold τ to very low value, such as 0.1, so that the label is not anymore part of the top 20 labels.

Labels (1-10)	Score	Labels (11-20)	Score
isolated	0.9945	corporate	0.9320
luggage	0.9921	one	0.9155
man	0.9727	casual	0.9132
case	0.9681	desktop	0.9131
briefcase	0.9672	executive	0.9116
business	0.9577	woman	0.9083
bag	0.9552	adult	0.90341
trip (journey)	0.9543	professional	0.9031
young	0.9484	fashion	0.8947
backpack	0.9325	elegant	0.8921

Table 4.6 Clarifai API results for `backpack.bmp`

Even some of the top 20 labels can be viewed as contradictory, e.g. ‘man’ vs. ‘woman’ and ‘business’ vs. ‘casual’. The results for each of the four images are summarized in the table below. The last row (Number of success tries with random noise out of 10) was obtained by running the method `compare_with_random_noise()`. The full results and Terminal outputs can be found in `report_results/clarifai_api`. The results can be reproduced as the subspace vectors used for each of the perturbed images is also saved in the above folder as a Pickle file with the name `<image_name>.p`. One can set the subspace vectors as a command line argument with `-v` in order to reproduce the results as such:

```
> %run fool_clarifai_api.py -f <image_path> -v <subspace_vectors_pickle_path>
```

Image in pics_bmp	backpack	crane-building	white-wolf-resize	cucumber_1
Targeted label	backpack	crane	wolf	cucumber
Original score	0.9325	0.9479	0.9412	0.9810
Removed labels	woman, backpack	tallest, crane, steel	tundra, wolf, dog, canine, polar, ice	cucumber
Added labels	people, person	technology, vehicle, daylight	merino, looking, one, rural, grass, sheep	fruit
Image norm	207.72261	271.70857	316.54845	149.85682
Perturbation norm	4.80558	15.19885	23.46766	2.59481
Number of successful tries with random noise out of 10	2	0	0	0

Table 4.7 Results for Clarifai API - Single Label Removal

4 RESULTS

The added noise is slightly more perceptible, particularly for `crane-building.bmp` and `white-wolf-resize.bmp`.



(a) Original, 0.9325 for 'backpack'



(b) Perturbed with 'backpack' removed

Figure 4.16 Binary DeepFool applied to Clarifai API and `pics_bmp/backpack.bmp`



(a) Original, 0.9479 for 'crane'



(b) Perturbed with 'crane' removed

Figure 4.17 Binary DeepFool applied to Clarifai API and `pics_bmp/crane-building.pdf.bmp`



(a) Original, 0.9412 for 'wolf'



(b) Perturbed with 'wolf' removed

Figure 4.18 Binary DeepFool applied to Clarifai API and `pics_bmp/white-wolf-resize.bmp`

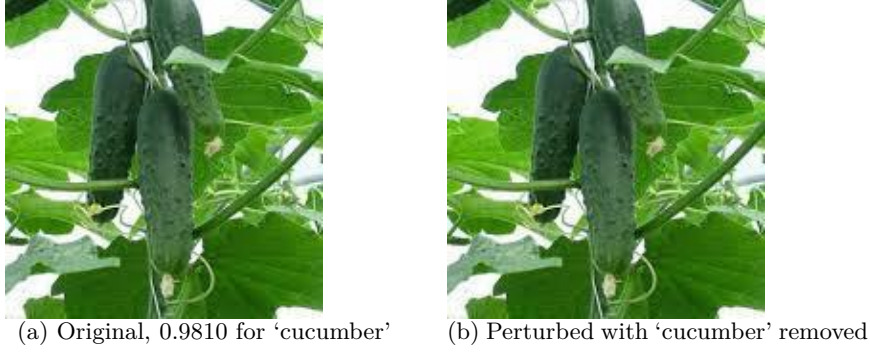


Figure 4.19 Binary DeepFool applied to Clarifai API and `pics_bmp/cucumber_1.bmp`

4.3.2 Multi-label removal

We also tried the Algorithm 3 variant of DeepFool in order to remove multiple labels. It was performed on `pics_bmp/cropped-panda.bmp` as seen in Figure 4.9. The original results can be seen below.

Labels (1-10)	Score	Labels (11-20)	Score
wildlife	0.9967	monkey	0.9395
cute	0.9947	portrait	0.9383
animal	0.9906	fur	0.9340
nature	0.9901	zoo	0.9326
mammal	0.9842	panda	0.9298
endangered species	0.9785	species	0.9210
little	0.9709	endangered	0.9145
sit	0.9647	outdoors	0.9139
looking	0.9615	downy	0.8880
wild	0.9398	primate	0.8790

Table 4.8 Original Clarifai API results for `cropped-panda.bmp`

We attempted removing the labels 'animal' and 'panda'. The result can be seen in Figure 4.20. We also tried removing just the label 'animal' to see if 'panda' would be removed as well. In the previous results for `white-wolf-resize`, we noticed that even though we just attempted to remove the label 'wolf', the labels 'dog' and 'canine' were also removed. Lo and behold, removing 'animal' also removed the label 'panda'. The results are summarized in the table below. The full results can be found in `report_results/clarifai_api` and be reproduced with the subspace vectors `report_results/clarifai_api/cropped-panda.p`.



Figure 4.20 Multi-Label and Single Label Removal DeepFool applied to Clarifai API and `pics_bmp/cropped_panda.bmp`

Task	Multi-label removal	Single label removal
Targeted label(s)	animal, panda	animal
Removed labels	species, downy, endangered species, endangered, primate, panda, sit, fur, zoo, little, wild, animal,	species, downy, endangered species, endangered, primate, panda, sit, fur, zoo, little, wild, animal
Added labels	people (new top label), young, facial expression, man, one, two, embrace, no person, eye, love, adult, face	people, young, facial expression, man, one, two, no person, woman, eye, side, view, adult, face
Image norm	85.63820	85.63820
Perturbation norm	8.98982	7.89407
Number of successful tries with random noise (out of 10)	0 for ‘animal’, 9 for ‘panda’	0 for ‘animal’, 4 for ‘panda’

Table 4.9 DeepFool results for Clarifai API and `cropped-panda.bmp`

Just removing ‘animal’ did a better job in that the perturbation norm is smaller so random noise of the same norm could not achieve the same results. This seems to suggest that certain labels appear together, but such a claim certainly requires a more in depth investigation. Perhaps multiple classifiers are at work within the Clarifai model - first to pick out general classes such as ‘food’ and ‘animal’. Consequently, a more detailed classifier may be used to identify the type of food or animal. It would definitely be interesting to investigate this (with more credits!).

5 Conclusion

Within a Python or IPython shell, one can conveniently apply DeepFool and analyze the results for the following classifiers: ResNet50, VGG16, VGG19, Inception v3, Clarifai, Google’s Cloud Vision, IBM’s Visual Recognition, Amazon Rekognition, and Microsoft’s Computer Vision. A few successful results have been observed for ResNet50, VGG16, VGG19, Inception v3, and Clarifai. The implementation has been made so that DeepFool can be easily extended to more classifiers.

I really enjoyed the experience of learning about the practical sides of machine learning, namely the state-of-the-art software such as TensorFlow, Theano, and Keras and the commercial products provided by some of the big names in industry. A lot of this software is still in the development stage or has been recently released so the documentation was not always the source of solutions. Thankfully, there is plenty of information on discussion boards and the developers are very quick to reply.

Future work

With the academic classifiers (ResNet50, VGG16, VGG19, Inception v3), it would be interesting to study *transferable* and *universal* adversarial perturbations [21][22], namely investigating whether a perturbed image could be computed *offline* on these academic networks to fool a blackbox classifier, such as one of the APIs. Another variant of DeepFool that could be implemented is fixed label perturbation, i.e. selecting a particular label to change the top label to.

Now that DeepFool is ready to use with several APIs (Clarifai, Google’s Cloud Vision, IBM’s Visual Recognition, Amazon Rekognition, and Microsoft’s Computer Vision), obtaining a subscription for more queries would allow for extensive testing. Moreover, extending to other API classifiers can be done with relative ease and very few lines of code, as the main components of DeepFool are already implemented.

To improve the results for the blackbox classifiers, it might be of use to investigate a different type of subspace vectors. The current ones consist of values drawn from the standard normal distribution (mean 0 and variance 1). Consequently, the vectors are constructed into an orthonormal basis. However, for typical image vector lengths ($299 \cdot 299 \cdot 3 = 268203$ for an input to Inception v3 which is already very small by image size standards), the individual components of the subspace vectors tend to be very small, as the norm of the vector will be 1. For this reason, we had to use a relatively high value for δ when estimating the gradients in order to cause a change in the image (as the pixel values would need to be quantized into integers within $[0, 255]$ when saved as an image). This differs from when working with the academic networks as we can deviate the input very slightly. When working with a blackbox classifier, however, we would not be able to have that large of a bit depth as to make such minuscule changes. In any case, the higher δ value did not result in an adversarial perturbation that was too noticeable. Perhaps a larger subspace dimension would also help in reducing the perturbation norm. This would, however, require a subscription in order to afford more queries.

Bibliography

- [1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, *et al.*, “Imagenet large scale visual recognition challenge”, *International Journal of Computer Vision (IJCV)*, vol. 115, pp. 211–252, 2015.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [3] M. A. Nielsen, *Neural Networks and Deep Learning*. New York: Determination Press, 2015.
- [4] www.clarifai.com.
- [5] www.tensorflow.org/tutorials/image_recognition/.
- [6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks”, *ArXiv preprint arXiv:1511.04599*, 2015.
- [7] github.com/LTS4/DeepFool.
- [8] cloud.google.com/vision/.
- [9] www.ibm.com/watson/developercloud/visual-recognition.html.
- [10] aws.amazon.com/rekognition/.
- [11] www.microsoft.com/cognitive-services/en-us/computer-vision-api.
- [12] github.com/tensorflow/tensorflow/blob/master/tensorflow/g3doc/api_docs/python/functions_and_classes/shard2/tf.test.compute_gradient.md.
- [13] goberoi.com/comparing-the-top-five-computer-vision-apis-98e3e3d7c647.
- [14] A. Fawzi, “Robust image classification: Analysis and applications”, PhD thesis, École Polytechnique Fédérale de Lausanne, 2016.
- [15] www.numpy.org/.
- [16] www.tensorflow.org.
- [17] deeplearning.net/software/theano/.
- [18] keras.io/.
- [19] www.tensorflow.org/versions/r0.10/tutorials/mnist/beginners/.
- [20] github.com/fchollet/deep-learning-models.
- [21] Y. Liu, X. Chen, Ch. Liu, and D. Song, “DELIVING INTO TRANSFERABLE ADVERSARIAL EX”,
- [22] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations”, *ArXiv preprint arXiv:1610.08401*, 2016.