Adversarial attack

April 10, 2018

1 Adversarial attacks on Google Inception V3 Image Classifier

1.1 Authors: Nisarg,Erin,Siva

1.2 Declarations

In [37]:
```
import torch
from torch import nn
from torch.autograd import Variable
from torch.autograd.gradcheck import zero_gradients
from torch.utils.data import Dataset, DataLoader

import torchvision.transforms as T
from torchvision.models.inception import inception_v3

from PIL import Image
from scipy.misc import imsave

import matplotlib.pyplot as plt
import os
import numpy as np
```

1.3 Reading classes and impoting the Inception model using Torch framework.

In [38]:
```
classes = eval(open('classes.txt').read())
trans = T.Compose([T.ToTensor(), T.Lambda(lambda t: t.unsqueeze(0))])
reverse_trans = lambda x: np.asarray(T.ToPILImage()(x))

eps = 2 * 8 / 225.
steps = 40
norm = float('inf')
step_alpha = 0.01

model = inception_v3(pretrained=True, transform_input=True).cpu()
loss = nn.CrossEntropyLoss()
model.eval();
```

1.4 Creating functions for Loading image, getting class details and Drawing the results

```
In [39]: def load_image(img_path):
             img = trans(Image.open(img_path).convert('RGB'))
             return img

         def get_class(img):
             x = Variable(img, volatile=True).cpu()
             cls = model(x).data.max(1)[1].cpu().numpy()[0]
             return classes[cls]

         def draw_result(img, noise, adv_img):
             fig, ax = plt.subplots(1, 3, figsize=(15, 10))
             orig_class, attack_class = get_class(img), get_class(adv_img)
             ax[0].imshow(reverse_trans(img[0]))
             ax[0].set_title('Original image: {}'.format(orig_class.split(',')[0]))
             ax[1].imshow(noise[0].cpu().numpy().transpose(1, 2, 0))
             ax[1].set_title('Attacking noise')
             ax[2].imshow(reverse_trans(adv_img[0]))
             ax[2].set_title('Adversarial example: {}'.format(attack_class))
             for i in range(3):
                 ax[i].set_axis_off()
             plt.tight_layout()
             plt.show()

         def resize_image(img):
             img = img.resize((299, 299), PIL.Image.ANTIALIAS)
             img.save('resized_image.jpg')
```

1.5 Creating method for Non-Targeted attack on DNN

1.5.1 The most general type of attack when all you want to do is to make the classifier give an incorrect result.

```
In [40]: def non_targeted_attack(img):
             #img = img.resize((299, 299), PIL.Image.ANTIALIAS)
             img = img.cpu()
             label = torch.zeros(1, 1).cpu()

             x, y = Variable(img, requires_grad=True), Variable(label)
             for step in range(steps):
                 zero_gradients(x)
                 out = model(x)
                 y.data = out.data.max(1)[1]
                 _loss = loss(out, y)
                 _loss.backward()
                 normed_grad = step_alpha * torch.sign(x.grad.data)
                 step_adv = x.data + normed_grad
```

```
            adv = step_adv - img
            adv = torch.clamp(adv, -eps, eps)
            result = img + adv
            result = torch.clamp(result, 0.0, 1.0)
            x.data = result
        return result.cpu(), adv.cpu()
```

1.6 Creating method for Targeted attack on DNN

1.6.1 This is slightly more difficult attack which aims to receive a particular class for your input.

```
In [41]: def targeted_attack(img, label):
            #img = img.resize((299, 299), PIL.Image.ANTIALIAS)
            img = img.cpu()
            label = torch.Tensor([label]).long().cpu()

            x, y = Variable(img, requires_grad=True), Variable(label)
            for step in range(steps):
                zero_gradients(x)
                out = model(x)
                _loss = loss(out, y)
                _loss.backward()
                normed_grad = step_alpha * torch.sign(x.grad.data)
                step_adv = x.data - normed_grad
                adv = step_adv - img
                adv = torch.clamp(adv, -eps, eps)
                result = img + adv
                result = torch.clamp(result, 0.0, 1.0)
                x.data = result
            return result.cpu(), adv.cpu()
```

2 Google Inception v3

2.1 We will demonstrate how a non-targeted adversarial attack can be used against Google's Inception v3 ImageNet classifier:

2.1.1 - A trained neural network, essentially, represents a high-dimensional decision boundary—think of it as a set of cells, where every point (in this case an image) in the same cell is associated with the same class. Of course, the boundary is not perfect—even more, if anything, these "cells" are too crude and linear, and that is their main vulnerability.

2.1.2 - Ideally, a good adversarial attack is a modified input that is visually indistinguishable from the original—yet the classifier gives a totally different prediction for it. The main idea behind it is to find a set of slight perturbations for every class of images that would "drag" the representation vector from the initial "cell" and put it into another. In this article, we will call the original image "Source" and the perturbation we add to it—"Noise". Although it's not really a noise, as we'll see, there's quite a lot of structure in it.

2.2 FGSM Algorithm (Fast Gradient Step Method)

2.2.1 - The core idea is to add some weak noise on every step of optimization, drifting towards the desired class—or, if you wish, away from the correct one. Sometimes we will have to limit the amplitude of noise to keep the attack subtle.

2.2.2 - for example, in case a human might be investigating our shenanigans. The amplitude in our case means the intensity of a pixel's channel—limiting it ensures that the noise will be almost imperceptible, and in the most extreme case will look like an overly compressed JPEG.

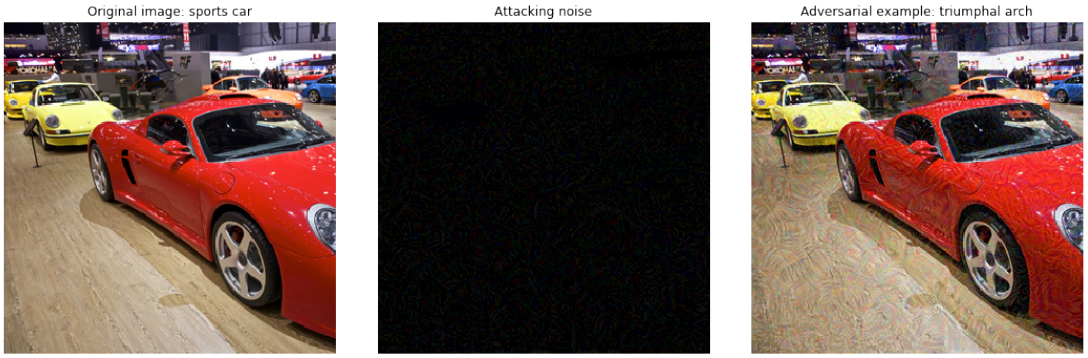
2.2.3 - This is a pure optimization problem—but in this case, we optimize the noise to maximize the error. You can directly measure the error and compute the gradient in this case since you have access to the raw outputs of the network.

2.2.4 - The border between "truth" and "false" is almost linear. The first cool thing we can derive from it is that when you follow the gradient, once you find the area where the predicted class changes, you can be fairly confident that the attack is successful. On the other hand, it tells us that the structure of the decision function is far simpler that most researchers thought it to be.

2.3 Generating and explaining a non-trivial exploit

```
In [6]: img = load_image('AS/1.png')
        adv_img, noise = non_targeted_attack(img)
        draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.3.1 The sports car was originally classified correctly bt after adding the noise it became Triumphal arch which is incorrect

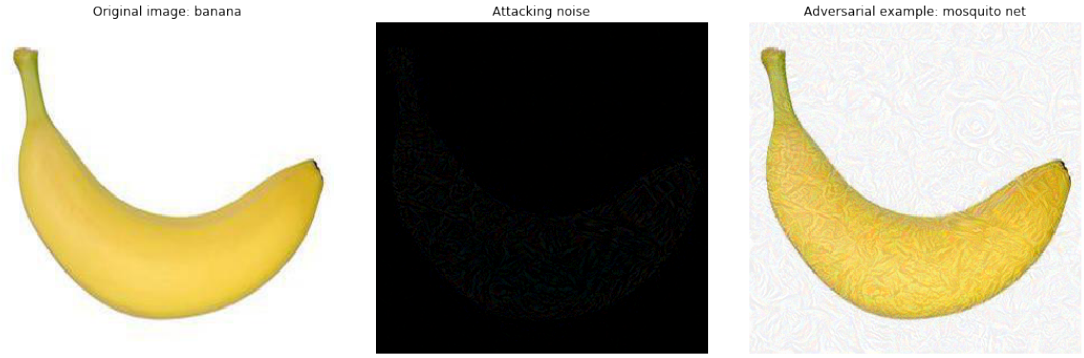
```
In [7]: img = load_image('AS/1.png')
        adv_img, noise = targeted_attack(img, 859)
        draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

2.3.2 The sports car was originally classified correctly but after adding the noise with target class value it became toaster which is incorrect

```
In [8]: img = load_image('AS/2.jpg')
        adv_img, noise = non_targeted_attack(img)
        draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

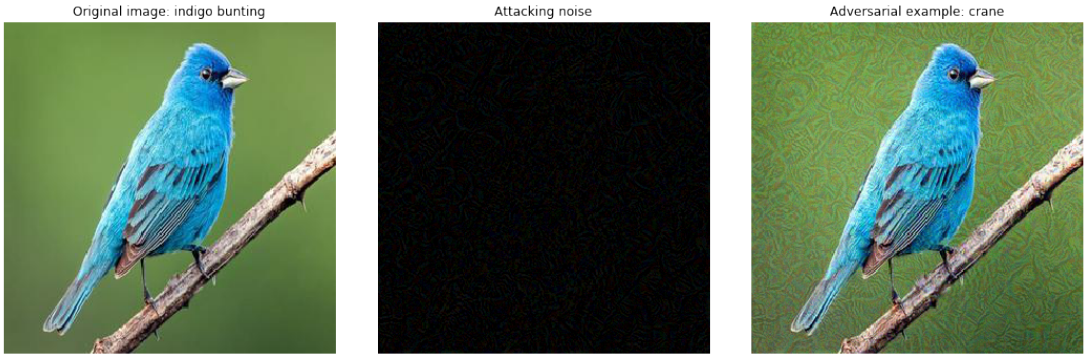


2.3.3 The banana was originally classified correctly but after adding the noise with target class value it became mosquito net which is incorrect

2.4 Contrastive examples. You must identify cases in which the system is operating as expected, and then alternative cases that appear equivalent

```
In [18]: img = load_image('AS/9.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.4.1 The Indigo bunting was originally classified correctly but after adding the noise with target class value it became Crane which is incorrect

```
In [10]: img = load_image('AS/3.jpeg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

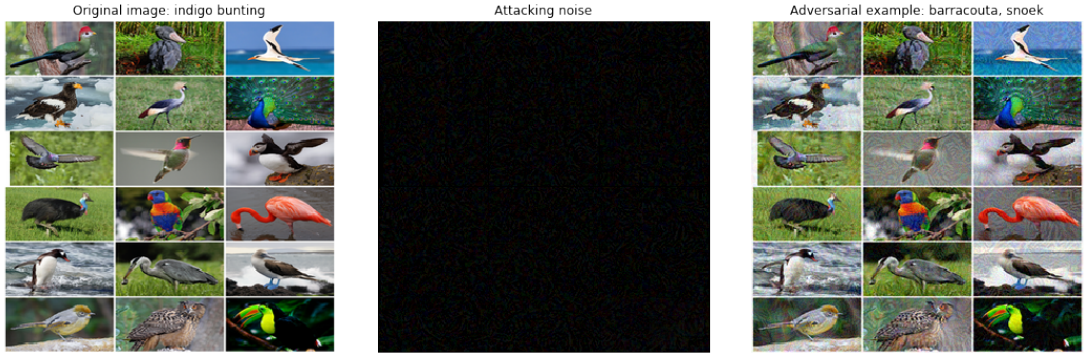
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

2.4.2 Results: The presented two cases for Indigo bunting is contrastive. The system is indetifying bird correctly in the first image, but in the second case it failed to identify "Group of Indigo bunting" Moreover, after adding noise in both the cases, for sure it misclassifies as always.

2.4.3 The same case for the grids of birds, system can only detect one type of bird successfully. Moreover, it fails after adding noise.

```
In [13]: img = load_image('AS/4.png')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

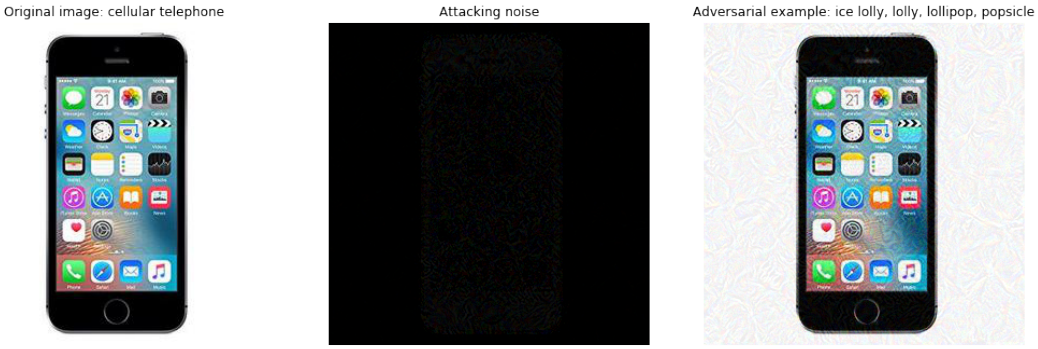
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.5 Demonstration of systematicity for the wrong predictions by DNN

```
In [14]: img = load_image('AS/5.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

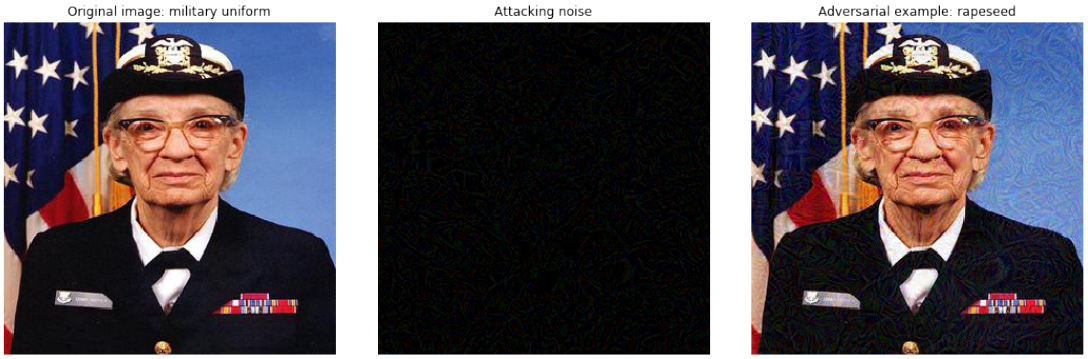
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.5.1 The cellular phone was originally classified correctly but after adding the noise it became Lollipop which is incorrect

```
In [15]: img = load_image('AS/6.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.5.2 The military uniform was originally classified correctly but after adding the noise it became rapeseed which is incorrect

2.6 Thus even for the known samples of image, the classifier performs very weird after noise addition. This systematic approach of adding noise in each gradient step is effective and we can repeat this and solve as a optimization problem where error is gonna maximize.

2.6.1 - take the known samples by classifier

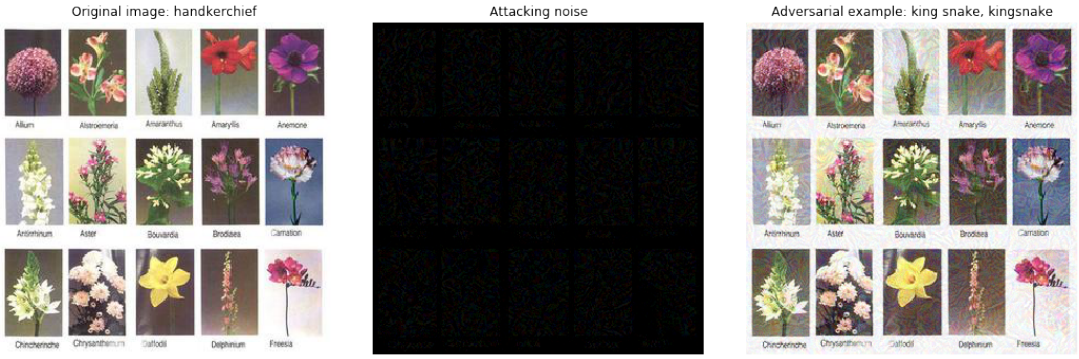
2.6.2 - Perform the noise addition using FGSM

2.6.3 - formulate the optimization problem and maximize the performance.

2.6.4 - feed in the adversarial image matrix and fool the DNN based classifiers

```
In [16]: img = load_image('AS/7.jpeg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.6.5 Results: The presented case of grid image of various type of flower was not identified by classifier. it was initially misclassified as hankerchief. Further, after adding noise it was obviously treated as unknown and classified as king snake.

```
In [17]: img = load_image('AS/8.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

2.6.6 Results: The presented case of grid image of various type of flower was not identified by classifier. it was initially misclassified as vase. Further, after adding noise it was obviously treated as unknown and classified as Wool.

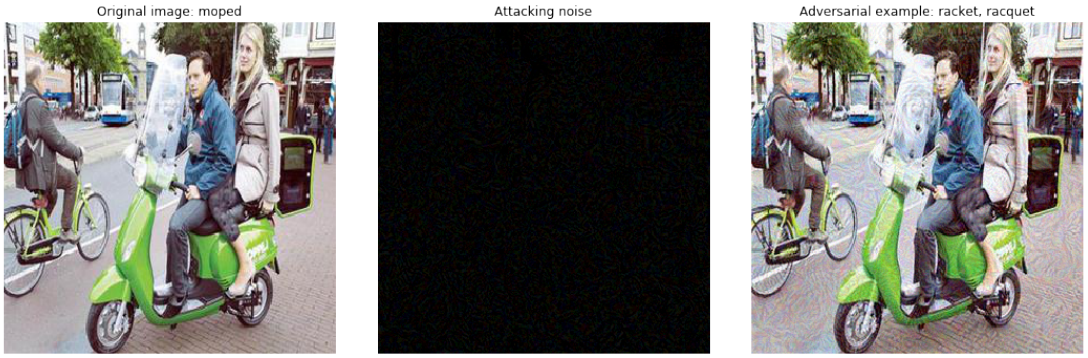
```
In [19]: img = load_image('AS/10.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

2.6.7 The motor scooter was originally classified correctly but after adding the noise it became daisy which is incorrect

```
In [20]: img = load_image('AS/11.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

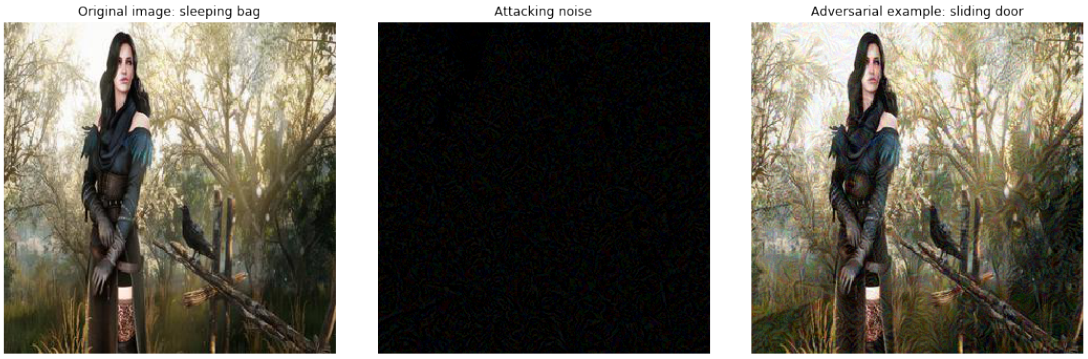
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.6.8 The moped was originally classified correctly but after adding the noise it became racket which is incorrect

```
In [21]: img = load_image('AS/12.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

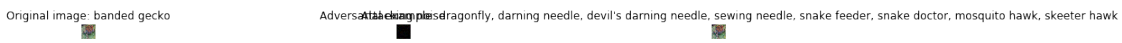
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.6.9 Results: The presented image of Yennefer (Character of Game Witcher) was not identified by classifier. it was initially misclassified as Sleeping bag. Further, after adding noise it was obviously treated as unknown and classified as Sliding door.

```
In [22]: img = load_image('AS/13.jpeg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

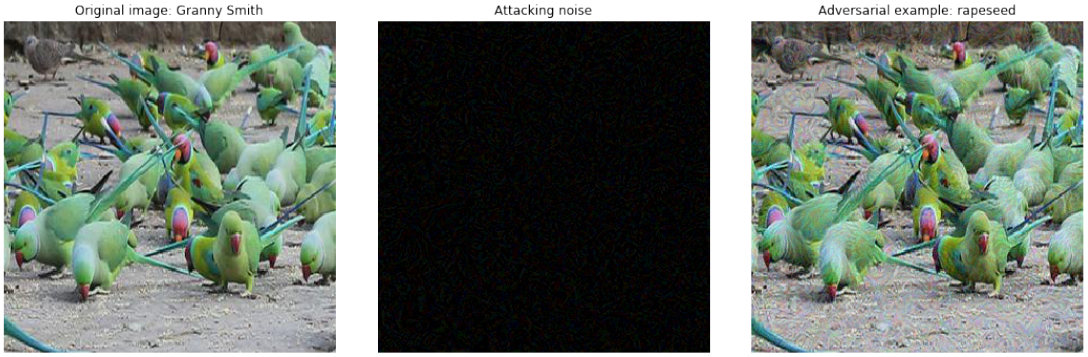
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.6.10 Results: The presented image of turtle was not identified by classifier. it was initially misclassified as Banded Gecko. Further, after adding noise it was obviously treated as unknown and classified as many vague things.

```
In [23]: img = load_image('AS/14.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



2.6.11 Results: The presented image of parrots was not identified by classifier. it was initially misclassified as Granny smith. Further, after adding noise it was obviously treated as unknown and classified as rapeseed.

2.7 An extreme outlier

2.7.1 Classifier doesn't know Kate Middleton! :P

```
In [24]: img = load_image('AS/15.jpg')
         adv_img, noise = non_targeted_attack(img)
         draw_result(img, noise, adv_img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```

2.7.2 Results: The presented image of kate Middleton was not identified by classifier. it was initially misclassified as Bearskin. Further, after adding noise it was obviously treated as unknown and classified as lakeland terrier.

3 Conclusion:

3.1 Using FGSM adversarial attack we can easily fool the neural network. We presented certain contrastive and trivial cases to prove that. Even for outliers it's definately the same case. Thus It's capable of fooling pretty much any machine learning algorithm in cases when there were no attempts to defend them.