Protecting networks against adversial attacks

Loïc Jézéquel

March 2019

1 Introduction

Neural networks are getting more and more used in our daily life for many different tasks leveraging computer vision. These kind of systems make some decision based on an image input. One of the most common task is *image classification*, and consists in finding the category in which the given image belongs to. For example, we could which to identify if a shape is a square, triangle or circle. In practice, we can find these systems used in many critical applications where the accuracy of the classification is essential to the overall security. Such applications range from filtering systems for uploaded images to autonomous car vision or CCTV...

However, because of the nature of these deployed networks, there exists a major flaw which can be used to deceive this classification, in order to control externally the system decisions without ever having to compromise the model but only the input images. This flaw is adversial learning. By using this method, we can transform an initial image of class A into an image that will most likely be classified by the system as another class B that we wish. These transformed images can be seen as optical illusions by analogy with the human vision system, but there are two major differences:

1. A human can't perceive any differences between the original and the transformed image. (as seen on figure 1)
2. The network is deceived by this image with very high confidence.

As we will see later, we need access to the neural network the classifier is using in order to generate such adversial images. However, with the increasing use of transfer learning and pretrained models in computer vision, the core of a lot of classification systems is commonly state of the art networks and openly available models. Therefore, images generated from one of this state of the art network can still highly deceive many commercialized systems.

In this article, we are going to explore ways to protect a classification network against such attack. However, in order to be able to use existing architectures without having to change it, we will only extend the training phase and add some preprocessing to the data.

2 Adversial learning

First of all, we are going to see how does adversial learning works and how adversial images can be produced.

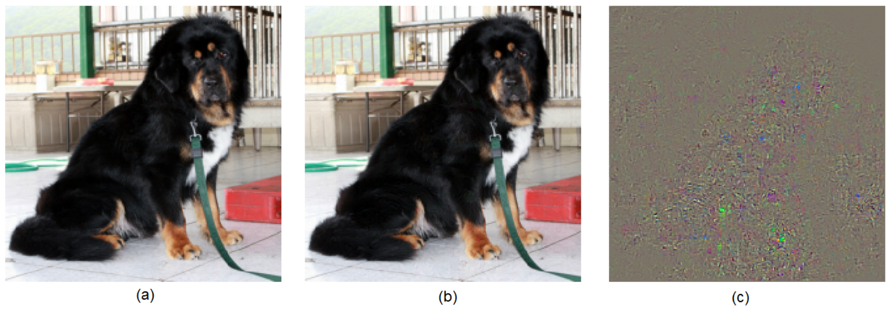

Figure 1: (a) The original image belonging to the dog class, (b) Transformed image classified as a goldfish, (c) The image difference amplified ten times

2.1 Principle

Let f be the classifier network we want to deceive, and $I = (x_1, ..., x_{W \times H})$ the input image. As previously stated $f(I) = c \in \{c_1, ..., c_N\}$, where c_i is the i^{th} object category. Adversial learning is an **iterative** process that will keep on modifying the original image until it is classified with high probability as the desired class c_{obj}. The loss function L used for the optimisation is the same as the regular loss function used for multi-class classification (usually cross-entropy loss) except instead of using the image real class label, we use c_{obj}. Then, to steer the maximum of class probability toward c_obj we perform gradient descent on the loss function considered in the space of the image.

To prevent over-changing the image, we normalize the gradient by its L_2 norm. Formally, the new image at each step is

$$I - \alpha \nabla_{x_1,...,x_{W \times H}} L\left(f(I^{(t)}), c_{obj}\right)$$

with α the update rate of the image.

The full algorithm of adversial learning is the following:

```
Set $c_{obj}$
Set $\alpha$

while f(I) $\neq$ $c_{obj}$ :
    $g \leftarrow \nabla_{x_1,...,x_{W \times H}} L\left(f(I^{(t)}), c_{obj}\right)$
    $I \leftarrow I - \alpha g / \|g\|$
```

2.2 Results

We tested adversial learning on three state of the art networks pretrained on the imagenet dataset. These networks are the **InceptionV3**, **VGG16** and **ResNet** which all are network with very high accuracy and commonly used.

We can notice that deeper networks such as VGG16 are harder to deceive than shallower networks as the inception network. Indeed, the difference starts to be noticable by a human.

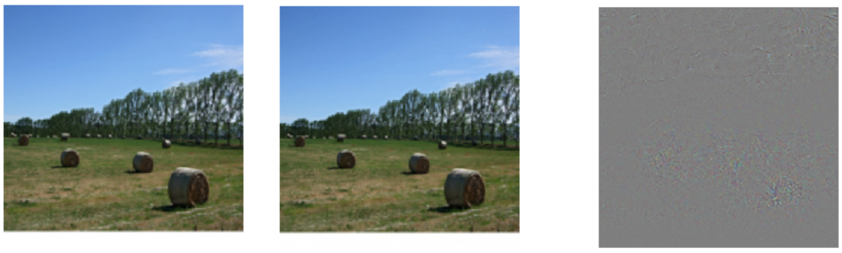
Figure 2: InceptionV3 : from Hay to Hen

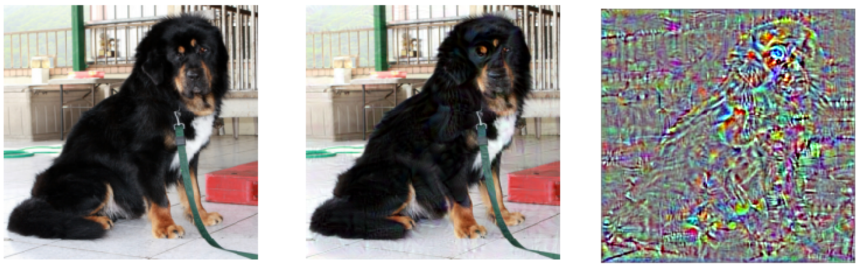
Figure 3: VGG16 : from Tibetan mastiff to kite

Here is the general python code to generate the adversial image:

```python
def get_adv_img(I, y_obj, model):

    I_adv = I.copy()

    # Update rate.
    update_rate = 0.1
    lr = tf.constant(update_rate, dtype="float32")

    # Loss function.
    loss = model.output[:, y_obj]
    g = tf.gradients(loss, model.input)
    g = tf.divide(tf.multiply(g, lr), tf.norm(g, ord=2))

    # Gradient descent.
    for step in range(1000):
      img_g, output = K.get_session().run([g, model.
          output], feed_dict={model.input:I_adv})
      img_g = np.array(img_g)
      I_adv += img_g[0]

      if output[0].argmax() == y_obj:
        break
    return I_adv
```

3 Anti-adversial learning

As stated in the introduction, we are here trying to derive learning methods to protect against adversial attacks. We are therefore mainly going to rely on preprocessing and postprocessing at training and evaluation.

Our main intuition is that pictures seen by a human can be interpreted as analogical data since we perceive continually lights emitted to our retina, while data seen by a computer system is always discrete. Thus, we could think preprocessing data to obtain evened-out data that could remove the effects of imperceptible modifications while still keeping good accuracy and details. The main method to even out our data is to apply some filter, in other words blur the images. Our "anti-adversial" learning would be the following: *at training time*, apply the preprocessing to the whole dataset so that the network learns using blurier versions of the objects. Then *at evaluation time*, we apply the exact same filter to the input image before feeding it to the network.

For the training, we will use a subset of the ImageNet dataset and use the recommended values for the optimisation hyperparameters except for the minibatch size which will be set to 64 because of memory limitations. As for the network, we will here only perform tests on the InceptionV5 model for time constraints.

The main challenge is now to choose which filter to apply, as the final network accuracy will greatly depend on how destructive the filter is. Therefore, we are going in the following sections to explore several filters with different hyperparameters, and look at their performances. The performances will be based on the accuracy and the norm of the difference between the original image and the adversial image. The best filter will be the one with the highest product (accuracy $\times \|I_{adv} - I\|$)

3.1 Bluring preprocessing

3.1.1 Gaussian filter

The first filter we are going to look at is the gaussian filter. It seems the more natural option as it is itself used in many approches to select the level of details to keep in an image (especially in scale-space approches). The gaussian kernel is defined as follow:

$$K(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

As we can see, σ can be interpreted as the extension of the filter, in other words the size of the window in which pixels will be mixed. σ will be an hyperparameter for the training and will directly deteriorate accuracy if too high.

We define the new training in python as follow:

```
def preprocess_data(X, sigma):
  X_pre = np.array([cv.GaussianBlur(img, (0,0), sigma)
      for img in X])
  return X_pre


for sigma in np.arange(1, 10, 0.2):
  train_dataset_pre = preprocess_data(train_dataset,
      sigma)
  valid_dataset_pre = preprocess_data(valid_dataset,
      sigma)

  model.compile(optimizer="adam", loss="
      categorical_crossentropy", metrics=["accuracy"])
  H = model.fit(train_dataset_pre, train_labels,
      validation_data=(valid_dataset_pre, valid_labels),
             epochs=500, batch_size=64)
```

3.1.2 Median filter

The gaussian kernel can be efficient to remove finite details, however it often ends up fading away edges which are essential to differentiate objects and keeping a good accuracy. Thus, we turn to the median filter which can remove small details while also preserving edges. The median filter is not a classical convolutional kernel, as it works by taking the median value of all the pixel contained in the current window. The size s of the window is once again a critical hyperparameter to the training.

The training using the median filter is as follow:

```
def preprocess_data(X, s):
  X_pre = np.array([cv.medianBlur(img, s) for img in X])
  return X_pre


for s in np.arange(3, 11, 2):
  train_dataset_pre = preprocess_data(train_dataset, s)
  valid_dataset_pre = preprocess_data(valid_dataset, s)

  model.compile(optimizer="adam", loss="
      categorical_crossentropy", metrics=["accuracy"])
  H = model.fit(train_dataset_pre, train_labels,
      validation_data=(valid_dataset_pre, valid_labels),
             epochs=500, batch_size=64)
```

However, in practice the gaussian kernel yields better accuracy. It might be because the median filter uniformizes textural areas which are important in some cases to tell apart two objects of the same shape. The best accuracy is acheived with a gaussian kernel and a $\sigma = 1.5$ (it is in fact near the smallest sigma we can use so that the filter has any effect).

4 Conclusion

We have been able to develop a learning method that can be applied to existing state of the art architectures in order to protect them against "adversial attacks". By utilising a gaussian filter, this method keeps good accuracy while producing very different adversial images.

Since this work is very far from perfect, there are many possible improvements and prospects. First of all, as training from scratch these large networks can take a lot of time; maybe transfer learning could have been used. However, leaving out the first layers of the retraining might lead to poor anti-adversial abilities since the basic filters would not be relearned.

Moreover, we have designed here an anti-adversial learning solely based on preprocessing and not modifying the network model. Designing a new network architecture that would maximize the distance between each class embeddings could be a direction to prevent adversial learning, however it is out of the scope of this article and would require replacing existing architectures which can be quite cumbersome for deployed systems.

To finish, I want to thank Mines Télécom that allowed me to stay in the cyber resilience laboratory leaded by Professor Kadobayashi in NAIST where I thought about this article. I am starting soon a 6 month internship to end my Master. I will be searching from October for a PhD topic in machine learning applied to computer vision to join this great adventure.

Loïc Jézéquel (loic.jezequel@telecom-sudparis.eu)