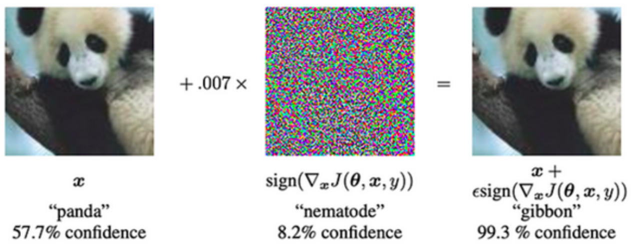
Adversarial example generation for face recognition model

Yuliia Hetman, juliagetman5@knu.ua

Abstract. In 2014, Goodfellow et al. published a paper entitled Explaining and Harnessing Adversarial Examples, which showed an intriguing property of deep neural networks — it's possible to purposely perturb an input image such that the neural network misclassified it. This type of perturbation is called an adversarial attack. This paper shows how adversarial attacks can be used to trick face recognition models. For this purpose we are going to implement a function which will calculate gradients of the cost function of the face recognition model. In this paper we are going to use a pre-trained keras model VGGFace2 to recognise celebrities' faces and Multi-task Cascaded Convolutional Neural Network (MTCNN) to detect them on images.

Adversarial attack. Adversarial attacks are a method of creating imperceptible changes to an image that can cause seemingly robust image classification techniques to misclassify an image consistently. Back in 2006, the top state-of-the-art machine learning models included Support Vector Machines (SVMs) and Random Forests (RFs) — it's been shown that both these types of models are susceptible to adversarial attacks.

With the rise in popularity of deep neural networks starting in 2012, it was hoped that these highly non-linear models would be less susceptible to attacks; however, Goodfellow et al. (among others) dashed these hopes.


+ .007 ×
=
x
"panda"
57.7% confidence
$\text{sign}(\nabla_x J(\theta, x, y))$
"nematode"
8.2% confidence
$x + \epsilon\text{sign}(\nabla_x J(\theta, x, y))$
"gibbon"
99.3 % confidence

The classic example of an adversarial attack can be seen above. On the left, we have our input image which our neural network correctly classifies as "panda" with 57.7% confidence. In the middle, we have a noise vector, which to the human eye, appears to be random. However, it's far from random. Instead, the pixels in the noise vector are equal to the sign of the elements of the gradient of the cost function with respect to the input image. Then this noise vector is added to the input image, which produces the output *(*right*)*. This image appears identical to the input; however, our neural network now classifies the image as a "gibbon" with 99.7% confidence.

Face detection. Before performing face recognition it is necessary to detect faces. Face detection is a process of automatically locating faces in a photograph and localizing them by drawing a bounding box around them.

To perform face detection Multi-task Cascaded Convolutional Neural Network (MTCNN) is being used. This is a state-of-the-art deep learning model for face detection, described in the 2016 paper titled "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks.". This neural network can be found in the package *mtcnn.*

The result of face detection using MTCNN face detector class is a list of bounding boxes, where each bounding box defines a lower-left-corner of the bounding box, as well as the width and height. Using these coordinates of the bounding boxes we crop the image.

Face Identification. To perform face recognition VGGFace2 model will be used. It is needed to install a third-party library for using VGGFace2 models in Keras which is called *keras-vggface*. A VGGFace model can be created using the *VGGFace()* constructor and specifying the type of model to create via the '*model*' argument. The *keras-vggface* library provides three pre-trained VGGModels, a VGGFace1 model via *model='vgg16'* (the default), and two VGGFace2 models '*resnet50*' and '*senet50*'. The first time that a model is created, the library will download the model weights and save them. The size of the weights for the *resnet50* model is about 158 megabytes.

Dataset of the pretrained model. The model expects input color images of faces with the shape of 244×244 and the output will be a class prediction of 8,631 people. This makes sense given that the pre-trained models were trained on 8,631 identities in the MS-Celeb-1M dataset. Unfortunately, readable labels of categories of the dataset were not found, that is why it was needed to preprocess a binary file with labels retrieving all categories and their indexes.

Non-adversarial face recognition. This Keras model can be used directly to predict the probability of a given face belonging to one or more of more than eight thousand known celebrities. Once a prediction is made, the class integers can be mapped to the names of the celebrities, and the top five names with the highest probability can be retrieved. This behavior is provided by the *decode_predictions()* function in the *keras-vggface* library.

The pixel values must be scaled in the same way that data was prepared when the VGGFace model was fit. Specifically, the pixel values must be centered on each channel using the mean from the training dataset. This can be achieved using the *preprocess_input()* function provided in the *keras-vggface* library and specifying the '*version=2*' so that the images are scaled using the mean values used to train the VGGFace2 models.

As a result the name of the celebrity on the photo will be predicted. This name will be used as a key to a particular index of categories. Here are the results of face recognition using the previous image of Sharon Stone face:

Name	**Confidence**
Sharon Stone	99.429%
Anita Lipnicka	0.223%
Bobbi Sue Luther	0.08%
Emma Atkins	0.029%
Noelle Reno	0.017%

Adversarial examples generation for face recognition. The function to generate the perturbation will take in an image to apply perturbations to, and the label that correctly classifies the image.

The *Sparse Categorical Cross-entropy* computes the categorical cross-entropy loss between the labels and predictions and will be used as a loss function. Generation of an adversarial example consists of several stages during iterating over steps:

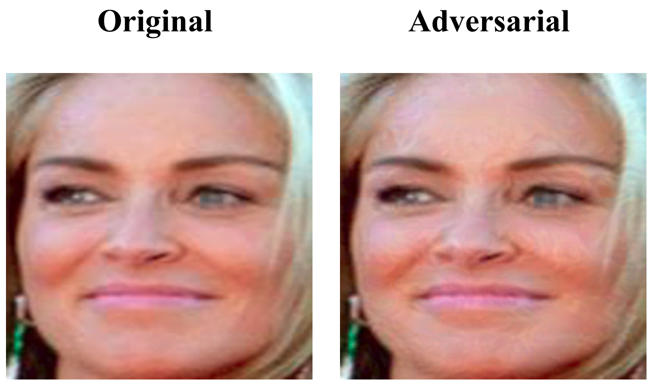
1. add perturbation vector to the base image and preprocess the resulting image;
2. run this newly constructed image tensor through the model and calculate the loss with respect to the original category index;
3. calculate the gradient of loss with respect to the perturbed vector;
4. update the weights, clip the perturbation vector and update the values.

By default we will take 50 steps to generate the adversarial example.

For the Sharon Stone image process of generating adversarials:

Steps	Loss
0	-0.006777158007025719
5	-0.10115277767181396
10	-2.4546256065368652
15	-7.9793829917907715
20	-12.94911003112793
25	-16.84220314025879
30	-19.839214324951172
35	-22.07464599609375
40	-23.84256362915039
45	-25.312740325927734

The resulting perturbation vector - the final noise vector - will allow us to construct the adversarial attack used to fool the model. Final adversarial image is constructed by adding the noise vector to the base image. To demonstrate that the function works, we create an adversarial image



After constructing an adversarial example, it's directed as an input for the face recognition model.

It can be seen that the predictions generated by VGGFace2 differ, because an adversarial example is incorrectly classified.

Name	Confidence
Bobbi Sue Luther	99.992156%
Maura Rivera	0.001006%
Sam Pinto	0.000491%
Dira Paes	0.000470%
Georgina Verbaan	0.000452%

Conclusions. Adversarial examples have attracted significant attention in machine learning. Researchers argue that there are microscopic features in every dataset that humans are unable to perceive, but neural networks utilize to their fullest extent. Adversarial examples manipulate these features to throw off classifiers, and they can do so without much effect to human viewers.

Code implementation.

https://github.com/yhetman/adversarial_example

In:

```
from numpy import expand_dims, asarray
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
import numpy as np
import json
import cv2
from mtcnn.mtcnn import MTCNN
from keras_vggface.vggface import VGGFace
from keras_vggface.utils import
preprocess_input, decode_predictions
from tensorflow.keras.optimizers import
Adam
from tensorflow.keras.losses import
SparseCategoricalCrossentropy

def extract_face(filename,
required_size=(224, 224)):
        print("[INFO] extracting only faces
from the image...")
        pixels = plt.imread(filename)
        detector = MTCNN()
        results =
detector.detect_faces(pixels)
        x1, y1, width, height =
results[0]['box']
        x2, y2 = x1 + width, y1 + height
        face = pixels[y1:y2, x1:x2]
        image = Image.fromarray(face)
        image = image.resize(required_size)
        face_array = asarray(image)
        return face_array

print("[INFO] loading image...")
pixels = extract_face('sharon_stone1.jpg')
pixels = pixels.astype('float32')

print("[INFO] expand dimensions...")
samples = expand_dims(pixels, axis=0)

print("[INFO] preprocess image...")
samples = preprocess_input(samples,
version=2)

print("[INFO] loading model...")
model = VGGFace(model='resnet50')

print("[INFO] making prediction...")
yhat = model.predict(samples)
```

```
print("[INFO] decoding predictions...")
results = decode_predictions(yhat)
for result in results[0]:
        print('%s: %.3f%%' %
(str(result[0].encode('utf-8').decode('utf-
8')), result[1]*100))
```

Out:

```
[INFO] loading image...
[INFO] extracting only faces from the
image...
[INFO] expand dimensions...
[INFO] preprocess image...
[INFO] loading model...
[INFO] making predictions...
[INFO] decoding predictions...
b' Sharon_Stone': 99.429%
b' Anita_Lipnicka': 0.223%
b' Bobbi_Sue_Luther': 0.080%
b' Emma_Atkins': 0.029%
b' Noelle_Reno': 0.017%
```

In:

```
from tensorflow.keras.applications.resnet50
import preprocess_input

with open('labels.json',"r+") as f:
        l = f.read()
data = json.loads(l)

def preprocess_image(image):
        image = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (224, 224))
        image = np.expand_dims(image, axis=0)
        return image

def clip_eps(tensor, eps):
        return tf.clip_by_value(tensor,
clip_value_min=-eps, clip_value_max=eps)


def generate_adversaries(model, baseImage,
delta, classIdx, steps=50):
        for step in range(0, steps):

        with tf.GradientTape() as tape:

                tape.watch(delta)
```

```
            adversary = preprocess_input(
baseImage + delta)


            predictions = model(adversary,
training=False)


            loss =
-sccLoss(tf.convert_to_tensor([classIdx]),
predictions)

            if step % 5 == 0:
            print("[INFO] step: {}, loss:
{}...".format(step, loss.numpy()))

            gradients =
tape.gradient(loss, delta)

optimizer.apply_gradients([(gradients,
delta)])
            delta =
delta.assign_add(clip_eps(delta, eps=EPS))
      return delta

EPS = 2 / 255
LR = 0.1

optimizer = Adam(learning_rate=LR)
sccLoss = SparseCategoricalCrossentropy()

print("[INFO] loading image...")
pixels = extract_face('sharon_stone1.jpg')
pixels = pixels.astype('float32')

print("[INFO] preprocessing image...")
image = preprocess_image(pixels)

print("[INFO] init constants and
variables...")
baseImage = tf.constant(image,
dtype=tf.float32)


delta =
tf.Variable(tf.zeros_like(baseImage),
trainable = True)

print("[INFO] generating adversarials
watching delta...")
deltaUpdated = generate_adversaries(model,
baseImage, delta, int(data['
Sharon_Stone']))

print("[INFO] creating adversarial
examples...")
adverImage = (baseImage +
deltaUpdated).numpy().squeeze()



adverImage = np.clip(adverImage, 0,
255).astype("uint8")



adverImage = cv2.cvtColor(adverImage,
cv2.COLOR_RGB2BGR)

print("[INFO] saving the adversarial
example...")
cv2.imwrite("adverImage.png", adverImage)
```

Out:

```
[INFO] loading image...
[INFO] extracting only faces from the
image...
[INFO] preprocessing image...
[INFO] init constants and variables...
[INFO] generating adversarials watching
delta...
[INFO] step: 0, loss:
-0.006777158007025719...
[INFO] step: 5, loss:
-0.10115277767181396...
[INFO] step: 10, loss:
-2.4546256065368652...
[INFO] step: 15, loss:
-7.9793829917907715...
[INFO] step: 20, loss:
-12.94911003112793...
[INFO] step: 25, loss:
-16.84220314025879...
[INFO] step: 30, loss:
-19.839214324951172...
[INFO] step: 35, loss:
-22.07464599609375...
[INFO] step: 40, loss:
-23.84256362915039...
[INFO] step: 45, loss:
-25.312740325927734...
[INFO] creating adversarial examples...
[INFO] saving the adversarial example...
```

In:

```
print("[INFO] reading adversarial image
examples...")
image = cv2.imread("adverImage.png")

print("[INFO] preprocessing adversarial
image example...")
image = preprocess_image(image)


preprocessedImage = preprocess_input(image)

print("[INFO] making predictions with
adversarial image examples...")
predictions =
model.predict(preprocessedImage)

print("[INFO] decoding predictions with
adversarial image examples...")
predictions =
decode_predictions(predictions)

for result in predictions[0]:
       print('%s: %.3f%%' %
(str(result[0].encode('utf-8').decode('utf-
8')), result[1]*100))
```

Out:

```
[INFO] reading adversarial image
examples...
[INFO] preprocessing adversarial image
example...
[INFO] making predictions with adversarial
image examples...
[INFO] decoding predictions with
adversarial image examples...
b' Bobbi_Sue_Luther': 99.992156%
b' Maura_Rivera': 0.001006%
b' Sam_Pinto': 0.000491%
b' Dira_Paes': 0.000470%
b' Georgina_Verbaan': 0.000452%
```