

Defending against adversarial image attacks with Keras and TensorFlow

(Phòng thủ chống lại các cuộc tấn công hình ảnh độc hại với Keras và TensorFlow.
)

Trong hướng dẫn này, bạn sẽ học cách phòng thủ trước các cuộc tấn công hình ảnh độc hại bằng cách sử dụng Keras và TensorFlow.

So far, you have learned how to generate adversarial images using three different methods:

- 1 [Adversarial images and attacks with Keras and TensorFlow](#)
- 2 [Targeted adversarial attacks with Keras and TensorFlow](#)
- 3 [Adversarial attacks with FGSM \(Fast Gradient Sign Method\)](#)

Bằng cách sử dụng hình ảnh gây thù hình, chúng ta có thể lừa các mạng Neural Convolutional (CNN) để đưa ra dự đoán sai. Trong khi đối với mắt người, hình ảnh gây thù có thể giống hệt với phiên bản gốc, chúng chứa những thay đổi nhỏ gây ra các dự đoán hoàn toàn sai lầm của các mạng CNN.

Như tôi đã thảo luận trong hướng dẫn này, có những hậu quả to lớn nếu triển khai các mô hình chưa được bảo vệ ra thực tế.

Ví dụ, hãy tưởng tượng một mạng neural sâu được triển khai trên một chiếc xe tự lái. Người dùng xấu có thể tạo ra hình ảnh gian lận, in chúng ra và sau đó áp dụng chúng lên đường, biển báo, cầu vượt, vv., dẫn đến mô hình nghĩ rằng có người đi bộ, xe hơi hoặc chướng ngại vật khi thực tế không có! Kết quả có thể rất nguy hiểm, bao gồm tai nạn xe hơi, chấn thương và mất mạng.

Given the risk that adversarial images pose, that raises the question:

“ What can we do to defend against these attacks?

Chúng tôi sẽ giải quyết câu hỏi đó trong một chuỗi hai phần về phòng thủ chống lại các cuộc tấn công hình ảnh đối kháng:

"Phòng thủ chống lại các cuộc tấn công hình ảnh đối kháng với Keras và TensorFlow" (hướng dẫn của hôm nay)

"Pha trộn hình ảnh bình thường và hình ảnh đối kháng khi huấn luyện CNNs" (hướng dẫn của tuần sau)

Phòng thủ chống lại các cuộc tấn công hình ảnh gian lận không phải là trò đùa. Nếu bạn triển khai các mô hình vào thế giới thực, hãy đảm bảo bạn có các quy trình để phòng thủ chống lại các cuộc tấn công hình ảnh gian lận.

Bằng cách làm theo các hướng dẫn này, bạn có thể huấn luyện mạng neural sâu của mình để đưa ra các dự đoán chính xác ngay cả khi chúng được trình bày với hình ảnh gian lận.

Để tìm hiểu cách huấn luyện một mạng neural tích chập để phòng thủ chống lại các cuộc tấn công hình ảnh gian lận với Keras và TensorFlow, chỉ cần tiếp tục đọc.

Defending against adversarial image attacks with Keras and TensorFlow

Trong phần đầu tiên của bài hướng dẫn này, chúng ta sẽ thảo luận về khái niệm của hình ảnh phản công như một "cuộc đua vũ khí" và những gì chúng ta có thể làm để bảo vệ chúng.

Chúng tôi sau đó sẽ thảo luận về hai phương pháp mà chúng ta có thể sử dụng để bảo vệ chống lại hình ảnh phản công. Chúng tôi sẽ triển khai phương pháp đầu tiên hôm nay và triển khai phương pháp thứ hai vào tuần sau.

Từ đó, chúng ta sẽ cấu hình môi trường phát triển của mình và xem xét cấu trúc thư mục dự án của chúng ta.

Sau đó, chúng ta sẽ xem xét một số tập lệnh Python, bao gồm:

1. Kiến trúc CNN của chúng ta
2. Một hàm được sử dụng để tạo ra hình ảnh gian lận bằng cách sử dụng FGSM
3. Một hàm tạo dữ liệu được sử dụng để tạo ra các lô hình ảnh gian lận để chúng ta có thể tinh chỉnh CNN của chúng ta trên chúng
4. Một tập lệnh đào tạo kết hợp tất cả các thành phần và đào tạo mô hình của chúng ta trên tập dữ liệu MNIST, tạo ra các hình ảnh gian lận, và sau đó tinh chỉnh CNN trên chúng để cải thiện độ chính xác

Let's get started!

Adversarial images are an “arms race,” and we need to defend against them



Figure 1: Defending against adversarial images is an arms race ([image source](#)).

Phòng thủ trước các cuộc tấn công phản đối đã và sẽ tiếp tục là một lĩnh vực nghiên cứu tích cực. Không có phương pháp "đạn đạo ma thuật" nào sẽ làm cho mô hình của bạn chống lại được các cuộc tấn công phản đối.

Thay vì đặt quan điểm của bạn về tấn công phản công — đó ít giống với một quy trình “đạn đạo ma thuật” và nhiều hơn là một cuộc chạy đua vũ trang.

Trong suốt Thế chiến thứ Hai giữa Hoa Kỳ và Liên Xô, cả hai nước đã bỏ ra số tiền khổng lồ và hàng giờ nghiên cứu và phát triển để cùng:

1. Chế tạo các vũ khí mạnh mẽ
2. Đồng thời tạo ra các hệ thống phòng thủ chống lại những vũ khí này.

Mỗi lần có sự di chuyển trên bàn cờ vũ khí hạt nhân đều có sự cố gắng bằng cách phòng thủ tương đương.

Chúng ta thường thấy những cuộc chạy đua vũ khí như vậy.

Một công ty tạo ra một sản phẩm mới trong ngành công nghiệp trong khi công ty khác tạo ra phiên bản riêng của họ. Một ví dụ tuyệt vời về điều này là Honda và Toyota. Khi Honda ra mắt Acura, phiên bản của họ về xe sang hơn vào năm 1986, Toyota đáp trả bằng việc tạo ra Lexus vào năm 1989, phiên bản của họ về xe sang.

Một ví dụ khác đến từ phần mềm chống virus, luôn luôn bảo vệ khỏi các cuộc tấn công mới. Khi một virus máy tính mới xuất hiện trên thế giới số, các công ty chống virus nhanh chóng phát hành các bản vá phần mềm để phát hiện và loại bỏ các virus này.

Cho dù chúng ta thích hay không, chúng ta sống trong một thế giới của sự leo thang liên tục. Với mỗi hành động, đều có một phản ứng tương đương. Đó không chỉ là vật lý, đó là cách thức của thế giới.

Nên không nên coi mô hình máy tính của chúng ta trong lĩnh vực thị giác máy tính và học sâu như là một không gian độc lập, không bị thao túng. Chúng có thể (và đang) bị thao túng.

Tương tự như máy tính của chúng ta có thể bị lây nhiễm bởi các loại virus được phát triển bởi hacker, các mạng neural của chúng ta cũng dễ bị tấn công bởi nhiều loại tấn công khác nhau, phổ biến nhất là tấn công đối kháng.

Thật may mắn là chúng ta có thể phòng thủ chống lại các cuộc tấn công này.

How can you defend against adversarial image attacks?

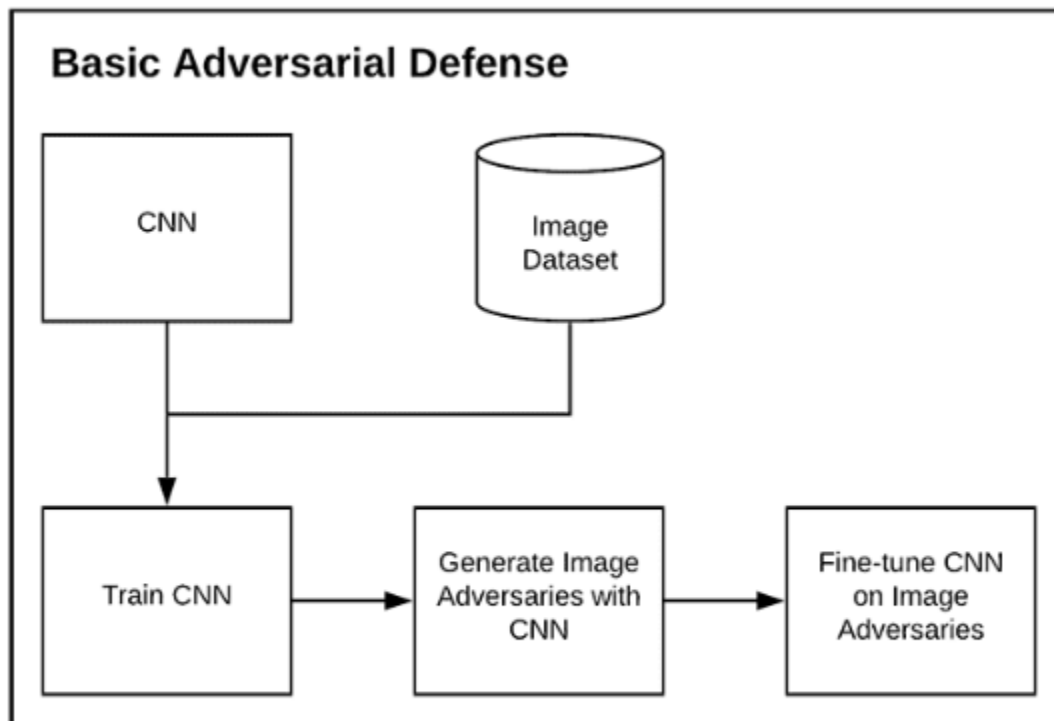


Figure 2: The process of training a model to defend against adversarial attacks.

Một trong những cách đơn giản nhất để phòng vệ chống lại các cuộc tấn công phản đối là huấn luyện mô hình của bạn trên các loại hình ảnh này.

Ví dụ, nếu chúng ta lo lắng người dùng xấu áp dụng tấn công FGSM vào mô hình của chúng ta, thì chúng ta có thể "tiêm chủng" mạng neural của mình bằng cách huấn luyện chúng trên các hình ảnh FGSM của riêng chúng ta.

Thường thì việc tiêm chủng phòng chống tấn công giả mạo như thế này được áp dụng bằng cách:

1. Huấn luyện mô hình của chúng ta trên một tập dữ liệu nhất định, tạo ra một tập hình ảnh gian lận, và sau đó tinh chỉnh mô hình trên các hình ảnh gian lận này.
2. Tạo ra các lô hỗn hợp gồm cả ảnh huấn luyện gốc và ảnh gian lận, sau đó tinh chỉnh mạng neural của chúng ta trên các lô hỗn hợp này.

Phương pháp đầu tiên đơn giản hơn và yêu cầu ít tính toán hơn (vì chúng ta chỉ cần tạo ra một tập hình ảnh gian lận). Nhược điểm là phương pháp này có xu hướng ít ổn định hơn vì chúng ta chỉ fine-tuning mô hình trên các ví dụ gian lận vào cuối quá trình huấn luyện.

Phương pháp thứ hai phức tạp hơn nhiều và yêu cầu tính toán nhiều hơn đáng kể. Chúng ta cần sử dụng mô hình để tạo ra các hình ảnh phản công cho mỗi lô dữ liệu được huấn luyện.

Lợi ích của phương pháp thứ hai là mô hình có xu hướng kháng lại hơn vì nó nhìn thấy cả ảnh huấn luyện gốc và ảnh gian lận trong mỗi lần cập nhật lô trong quá trình huấn luyện.

Đồng thời, mô hình sẽ được sử dụng để tạo ra các hình ảnh gian lận trong mỗi batch. Khi mô hình càng tốt hơn trong việc tự đánh lừa mình, nó cũng có thể học từ những sai lầm của mình, dẫn đến một mô hình có thể phòng thủ tốt hơn trước các cuộc tấn công gian lận

Chúng ta sẽ đề cập đến phương pháp đầu tiên ở đây hôm nay. Tuần sau, chúng ta sẽ thực hiện phương pháp nâng cao hơn.

Problems and considerations with adversarial image defense

(Vấn đề và những yếu tố cần xem xét khi bảo vệ chống lại tấn công ảnh gian lận)

Cả hai phương pháp phòng thủ ảnh gian lận được đề cập trong phần trước đó phụ thuộc vào:

1. Kiến trúc mô hình và trọng số được sử dụng để tạo ra các ví dụ ảnh gian lận
2. Bộ tối ưu hóa được sử dụng để tạo ra chúng

Các phương pháp đào tạo này có thể không tổng quát tốt nếu chúng ta đơn giản chỉ tạo ra một hình ảnh phản kích với một mô hình khác (có thể là một mô hình phức tạp hơn).

Thêm vào đó, nếu chúng ta chỉ huấn luyện trên các hình ảnh phản đối thủ địch thì mô hình có thể không hoạt động tốt trên các hình ảnh thông thường. Hiện tượng

này thường được gọi là quên đột ngột, và trong ngữ cảnh phòng thủ phản đối thù địch, có nghĩa là mô hình đã "quên" đi hình ảnh thực sự như thế nào.

Để giảm thiểu vấn đề này, chúng ta trước tiên tạo ra một tập hình ảnh gian lận, pha trộn chúng với tập huấn luyện thông thường, và sau đó mới huấn luyện mô hình (mà chúng ta sẽ thực hiện trong bài đăng của tuần tới).

Configuring your development environment

Hướng dẫn này về cách phòng chống tấn công hình ảnh gian lận sử dụng Keras và TensorFlow. Nếu bạn định theo dõi hướng dẫn này, tôi đề xuất bạn dành thời gian cấu hình môi trường phát triển học sâu của mình.

Bạn có thể sử dụng bất kỳ trong hai hướng dẫn sau để cài đặt TensorFlow và Keras trên hệ thống của bạn:

Cách cài đặt TensorFlow 2.0 trên Ubuntu

Cách cài đặt TensorFlow 2.0 trên macOS

Bất kỳ hướng dẫn nào cũng sẽ giúp bạn cấu hình hệ thống của mình với tất cả các phần mềm cần thiết cho bài đăng trên blog này trong một môi trường Python ảo thuận tiện.

Having problems configuring your development environment?

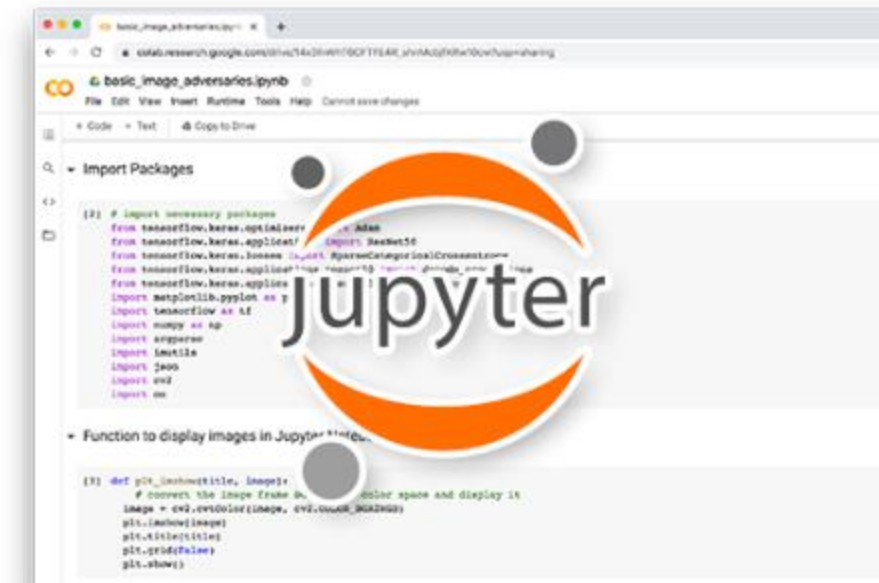


Figure 3: Having trouble configuring your dev environment? Want access to pre-configured Jupyter Notebooks running on Google Colab? Be sure to join [PyImageSearch University](#) — you'll be up and running with this tutorial in a matter of minutes.

Tất cả những gì được nói trên, bạn có:

1. Ít thời gian?
2. Học trên hệ thống bị khóa quản trị của nhà tuyển dụng của bạn?
3. Muốn bỏ qua rắc rối khi đối phó với dòng lệnh, trình quản lý gói và môi trường ảo?
4. Sẵn sàng chạy mã nguồn ngay bây giờ trên các hệ thống Windows, macOS hoặc Linux của bạn?

Project structure

Before we dive into any code, let's first review our project directory structure.

Be sure to access the “Downloads” section of this guide to retrieve the source code:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
1. $ tree . --dirsfirst
2.
3. ├── pyimagesearch
4. │   ├── __init__.py
5. │   ├── datagen.py
6. │   ├── fgsm.py
7. │   └── simplecnn.py
8. └── train_adversarial_defense.py
9.
10. 1 directory, 5 files
```

Trong module `pyimagesearch`, bạn sẽ tìm thấy ba tệp tin:

`datagen.py`: Thực hiện hàm để tạo ra các batch hình ảnh gian lận một lần. Chúng ta sẽ sử dụng hàm này để huấn luyện và đánh giá CNN của mình về độ chính xác trong việc phòng thủ ảnh gian lận.

`fgsm.py`: Thực hiện phương pháp Fast Gradient Sign Method (FGSM) để tạo ra hình ảnh gian lận.

`simplecnn.py`: Kiến trúc CNN của chúng ta sẽ được huấn luyện và đánh giá để phòng thủ ảnh gian lận.

Cuối cùng, `train_adversarial_defense.py` kết nối tất cả các phần lại với nhau và sẽ thể hiện:

Cách để huấn luyện kiến trúc CNN của chúng ta

Cách để đánh giá CNN của chúng ta trên tập kiểm tra

Cách để tạo ra các batch hình ảnh tấn công bằng cách sử dụng CNN đã được huấn luyện của chúng ta

Cách để đánh giá độ chính xác của CNN của chúng ta trên các hình ảnh tấn công

Cách để tinh chỉnh lại CNN của chúng ta trên các hình ảnh tấn công

Cách để đánh giá lại CNN của chúng ta trên cả tập huấn luyện gốc và các hình ảnh tấn công

Tới cuối hướng dẫn này, bạn sẽ có được một hiểu biết

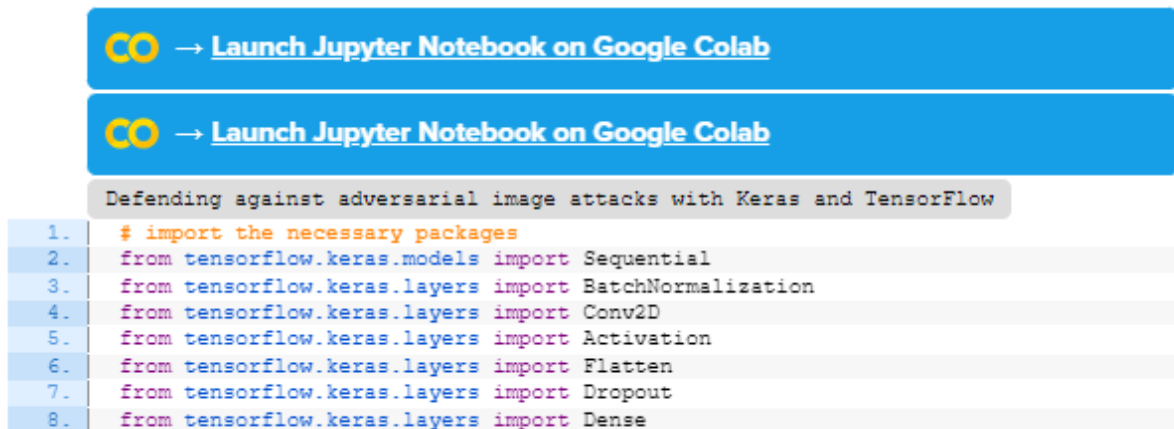
tốt về cách huấn luyện một CNN để phòng thủ cơ bản trước các hình ảnh tấn công.

Our simple CNN architecture

Chúng ta sẽ huấn luyện một kiến trúc CNN cơ bản và sử dụng nó để thể hiện phòng thủ hình ảnh đối kháng.

Mặc dù tôi đã bao gồm thực hiện kiến trúc này ở đây hôm nay, tôi đã đề cập đến kiến trúc chi tiết trong hướng dẫn tuần trước về Phương pháp Dấu hiệu Gradient Nhanh, vì vậy tôi đề xuất bạn tham khảo đó nếu bạn cần một bài đánh giá toàn diện hơn.

Mở tệp `simplecnn.py` trong mô-đun `pyimagesearch` của bạn, và bạn sẽ tìm thấy mã sau đây:



The screenshot shows a Jupyter Notebook interface with two blue buttons at the top, each with a yellow 'CO' icon and the text '→ Launch Jupyter Notebook on Google Colab'. Below the buttons is a code cell with the title 'Defending against adversarial image attacks with Keras and TensorFlow'. The code cell contains eight lines of Python code for importing necessary packages:

```
1. # import the necessary packages
2. from tensorflow.keras.models import Sequential
3. from tensorflow.keras.layers import BatchNormalization
4. from tensorflow.keras.layers import Conv2D
5. from tensorflow.keras.layers import Activation
6. from tensorflow.keras.layers import Flatten
7. from tensorflow.keras.layers import Dropout
8. from tensorflow.keras.layers import Dense
```

Phía trên tệp của chúng ta bao gồm các lệnh import Keras và TensorFlow.

Sau đó, chúng ta định nghĩa kiến trúc SimpleCNN.

CO → [Launch Jupyter Notebook on Google Colab](#)

CO → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
10. class SimpleCNN:
11.     @staticmethod
12.     def build(width, height, depth, classes):
13.         # initialize the model along with the input shape
14.         model = Sequential()
15.         inputShape = (height, width, depth)
16.         chanDim = -1
17.
18.         # first CONV => RELU => BN layer set
19.         model.add(Conv2D(32, (3, 3), strides=(2, 2), padding="same",
20.             input_shape=inputShape))
21.         model.add(Activation("relu"))
22.         model.add(BatchNormalization(axis=chanDim))
23.
24.         # second CONV => RELU => BN layer set
25.         model.add(Conv2D(64, (3, 3), strides=(2, 2), padding="same"))
26.         model.add(Activation("relu"))
27.         model.add(BatchNormalization(axis=chanDim))
28.
29.         # first (and only) set of FC => RELU layers
30.         model.add(Flatten())
31.         model.add(Dense(128))
32.         model.add(Activation("relu"))
33.         model.add(BatchNormalization())
34.         model.add(Dropout(0.5))
35.
36.         # softmax classifier
37.         model.add(Dense(classes))
38.         model.add(Activation("softmax"))
39.
40.         # return the constructed network architecture
41.         return model
```

Thực chất, hàm này theo dõi gradient của hình ảnh, đưa ra dự đoán, tính toán tổn thất, sau đó sử dụng dấu của gradient để cập nhật độ sáng của các pixel đầu vào, sao cho:

Hình ảnh cuối cùng được phân loại sai bởi mô hình CNN của chúng ta Tuy nhiên, hình ảnh trông giống như ban đầu (theo đánh giá của con người)

Xin tham khảo bài hướng dẫn tuần trước về Fast Gradient Sign Method để biết thêm chi tiết về cách hoạt động của kỹ thuật này và cách thực hiện nó.

Implementing a custom data generator used to generate adversarial images during training

Việc thực hiện bộ tạo dữ liệu tùy chỉnh được sử dụng để tạo ra các hình ảnh thù địch trong quá trình đào tạo

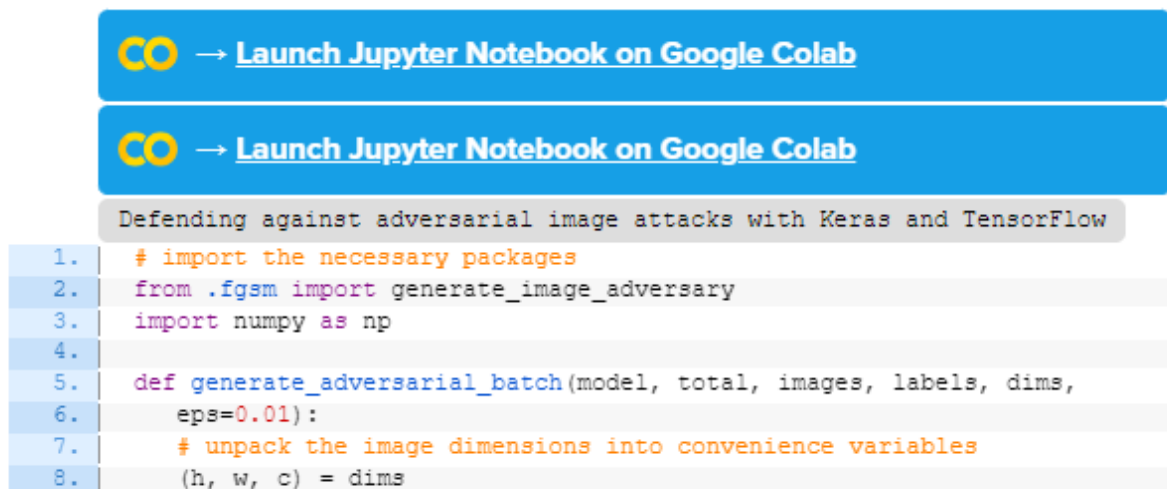
Hàm quan trọng nhất của chúng ta ở đây là phương thức `generate_adversarial_batch`. Hàm này là bộ tạo dữ liệu tùy chỉnh mà chúng tôi sẽ sử dụng trong quá trình đào tạo.

Ở mức độ cao, hàm này:

Chấp nhận một tập hợp các hình ảnh đào tạo Lấy ngẫu nhiên một lô kích thước N từ các hình ảnh đào tạo của chúng ta

Áp dụng `generate_image_adversary` để tạo ra hình ảnh thù địch Trả về lô hình ảnh thù địch cho vòng lặp đào tạo của chúng tôi, cho phép mô hình học các mẫu từ hình ảnh thù địch và đối phó với chúng trong tương lai.

chúng ta hãy xem vào hàm tạo dữ liệu tùy chỉnh của chúng ta. Mở file `datagen.py` trong thư mục dự án của chúng ta và thêm mã sau đây:



```
1. # import the necessary packages
2. from .fgsm import generate_image_adversary
3. import numpy as np
4.
5. def generate_adversarial_batch(model, total, images, labels, dims,
6.     eps=0.01):
7.     # unpack the image dimensions into convenience variables
8.     (h, w, c) = dims
```

Bắt đầu bằng cách nhập các gói phần mềm cần thiết. Lưu ý rằng chúng ta đang sử dụng triển khai FGSM của chúng tôi thông qua chức năng `generate_image_adversary` mà chúng ta đã triển khai trước đó.

Phương thức `generate_adversarial_batch` của chúng ta yêu cầu một vài tham số bao gồm:

model: CNN mà chúng ta muốn đánh lừa (tức là, mô hình chúng ta đang huấn luyện).

total: Kích thước của lô các hình ảnh gian lận chúng ta muốn tạo ra.

images: Tập hình ảnh mà chúng ta sẽ lấy mẫu từ (thông thường là tập huấn luyện hoặc tập kiểm tra).

labels: Các nhãn lớp tương ứng cho các hình ảnh.

dims: Kích thước không gian của các hình ảnh đầu vào.

eps: Một yếu tố epsilon nhỏ được sử dụng để điều khiển độ lớn của cập nhật độ sáng của điểm ảnh khi áp dụng phương pháp Fast Gradient Sign Method

Dòng 8 giải nén dims thành chiều cao (h), chiều rộng (w) và số kênh (c) để chúng ta có thể dễ dàng tham chiếu đến chúng trong phần còn lại của hàm.

Let's now build the data generator itself:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
10.     # we're constructing a data generator here so we need to loop
11.     # indefinitely
12.     while True:
13.         # initialize our perturbed images and labels
14.         perturbImages = []
15.         perturbLabels = []
16.
17.         # randomly sample indexes (without replacement) from the
18.         # input data
19.         idxs = np.random.choice(range(0, len(images)), size=total,
20.                                replace=False)
```

Điều kiện trong vòng lặp while ở dòng 12 sẽ chạy vô hạn cho đến khi quá trình huấn luyện kết thúc.

Tiếp theo, chúng ta khởi tạo hai danh sách perturbImages (để lưu trữ các hình ảnh gian lận được tạo ra trong lô sau này) và perturbLabels (để lưu trữ các nhãn lớp ban đầu cho hình ảnh).

Dòng 19 và 20 lấy một tập hợp ngẫu nhiên các hình ảnh của chúng ta.

Bây giờ chúng ta có thể lặp lại các chỉ mục của mỗi hình ảnh được chọn ngẫu nhiên:

```
CO → Launch Jupyter Notebook on Google Colab

CO → Launch Jupyter Notebook on Google Colab

Defending against adversarial image attacks with Keras and TensorFlow

22.     # loop over the indexes
23.     for i in idxs:
24.         # grab the current image and label
25.         image = images[i]
26.         label = labels[i]
27.
28.         # generate an adversarial image
29.         adversary = generate_image_adversary(model,
30.             image.reshape(1, h, w, c), label, eps=eps)
31.
32.         # update our perturbed images and labels lists
33.         perturbImages.append(adversary.reshape(h, w, c))
34.         perturbLabels.append(label)
35.
36.         # yield the perturbed images and labels
37.         yield (np.array(perturbImages), np.array(perturbLabels))
```

Các dòng 25 và 26 lấy ảnh và nhãn hiện tại.

Chúng ta sau đó áp dụng hàm `generate_image_adversary` để tạo ra ảnh phản đối với FGSM (dòng 29 và 30).

Sau khi tạo ra ảnh phản đối, chúng tôi cập nhật cả hai danh sách `perturbImages` và `perturbLabels`.

Cuối cùng, generator của chúng tôi đưa ra một 2-tuple của các ảnh và nhãn phản đối cho quá trình huấn luyện.

Hàm này có thể được tóm tắt như sau:

- Nhận một tập hình ảnh đầu vào

- Ngẫu nhiên chọn một phần của chúng

- Tạo ra các ảnh tấn công cho từng phần

Trả về các ảnh tấn công cho quá trình huấn luyện, để mô hình CNN của chúng ta có thể học các mẫu từ chúng.

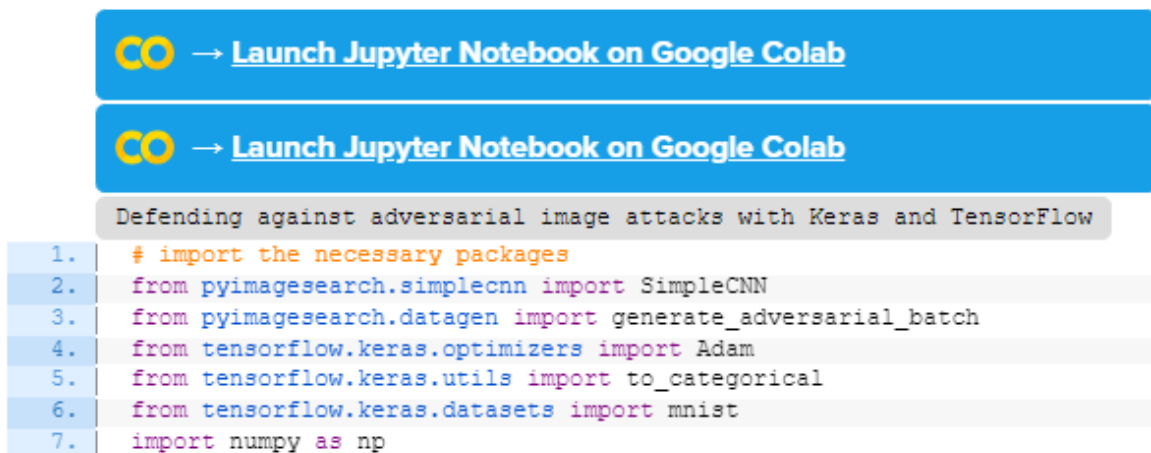
Điều này đúng. Nếu chúng ta huấn luyện CNN trên cả các hình ảnh huấn luyện gốc và các hình ảnh đối thủ, thì CNN của chúng ta có thể đưa ra các dự đoán chính xác trên cả hai tập hình ảnh, từ đó làm cho mô hình của chúng ta chống lại các cuộc tấn công đối thủ mạnh hơn.

Training on normal images, fine-tuning on adversarial images

(Huấn luyện trên ảnh bình thường, tinh chỉnh trên ảnh đối kháng)

Với tất cả các hàm trợ giúp của chúng ta đã được triển khai, hãy tiếp tục tạo mã đào tạo của chúng tôi để bảo vệ chống lại hình ảnh phản công.

Mở tệp `train_adversarial_defense.py` trong cấu trúc dự án của bạn và bắt đầu làm việc.



```
CO → Launch Jupyter Notebook on Google Colab
CO → Launch Jupyter Notebook on Google Colab
Defending against adversarial image attacks with Keras and TensorFlow
1. # import the necessary packages
2. from pyimagesearch.simplecnn import SimpleCNN
3. from pyimagesearch.datagen import generate_adversarial_batch
4. from tensorflow.keras.optimizers import Adam
5. from tensorflow.keras.utils import to_categorical
6. from tensorflow.keras.datasets import mnist
7. import numpy as np
```

Các dòng 2-7 trong mã nguồn thực hiện việc import các gói phần mềm Python cần thiết. Lưu ý rằng chúng ta đã import kiến trúc SimpleCNN cùng với hàm `generate_adversarial_batch` mà chúng ta mới thực hiện.

Tiếp theo, chúng ta tiến hành tải tập dữ liệu MNIST và tiền xử lý dữ liệu:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
9. # load MNIST dataset and scale the pixel values to the range [0, 1]
10. print("[INFO] loading MNIST dataset...")
11. (trainX, trainY), (testX, testY) = mnist.load_data()
12. trainX = trainX / 255.0
13. testX = testX / 255.0
14.
15. # add a channel dimension to the images
16. trainX = np.expand_dims(trainX, axis=-1)
17. testX = np.expand_dims(testX, axis=-1)
18.
19. # one-hot encode our labels
20. trainY = to_categorical(trainY, 10)
21. testY = to_categorical(testY, 10)
```

Sau khi đã tải tập dữ liệu MNIST, chúng ta có thể biên dịch mô hình và huấn luyện nó trên tập huấn luyện của chúng ta.

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
23. # initialize our optimizer and model
24. print("[INFO] compiling model...")
25. opt = Adam(lr=1e-3)
26. model = SimpleCNN.build(width=28, height=28, depth=1, classes=10)
27. model.compile(loss="categorical_crossentropy", optimizer=opt,
28.               metrics=["accuracy"])
29.
30. # train the simple CNN on MNIST
31. print("[INFO] training network...")
32. model.fit(trainX, trainY,
33.           validation_data=(testX, testY),
34.           batch_size=64,
35.           epochs=20,
36.           verbose=1)
```


Bước tiếp theo là đánh giá mô hình trên tập kiểm tra:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
38. # make predictions on the testing set for the model trained on
39. # non-adversarial images
40. (loss, acc) = model.evaluate(x=testX, y=testY, verbose=0)
41. print("[INFO] normal testing images:")
42. print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
43.
44. # generate a set of adversarial from our test set
45. print("[INFO] generating adversarial examples with FGSM...\n")
46. (advX, advY) = next(generate_adversarial_batch(model, len(testX),
47. testX, testY, (28, 28, 1), eps=0.1))
48.
49. # re-evaluate the model on the adversarial images
50. (loss, acc) = model.evaluate(x=advX, y=advY, verbose=0)
51. print("[INFO] adversarial testing images:")
52. print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
```

Các dòng 40-42 sử dụng mạng CNN đã được huấn luyện để đưa ra dự đoán trên tập kiểm tra. Sau đó, chúng tôi hiển thị độ chính xác và tổn thất trên terminal.

Bây giờ, hãy xem hiệu suất của mô hình của chúng tôi trên các hình ảnh phản đối.

Dòng 46 và 47 tạo ra một tập hình ảnh phản đối trong khi các dòng 50-52 đánh giá lại mạng CNN đã được huấn luyện của chúng ta trên các hình ảnh phản đối này. Như chúng ta sẽ thấy trong phần tiếp theo, độ chính xác dự đoán của chúng tôi giảm đáng kể trên các hình ảnh phản đối.

Điều đó đặt ra câu hỏi:

How can we defend against these adversarial attacks?

Làm thế nào chúng ta có thể bảo vệ chống lại các cuộc tấn công phản đối ảnh?

Một giải pháp cơ bản là tinh chỉnh mô hình của chúng ta trên các hình ảnh phản đối:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
54. # lower the learning rate and re-compile the model (such that we can
55. # fine-tune it on the adversarial images)
56. print("[INFO] re-compiling model...")
57. opt = Adam(lr=1e-4)
58. model.compile(loss="categorical_crossentropy", optimizer=opt,
59.               metrics=["accuracy"])
60.
61. # fine-tune our CNN on the adversarial images
62. print("[INFO] fine-tuning network on adversarial examples...")
63. model.fit(advX, advY,
64.           batch_size=64,
65.           epochs=10,
66.           verbose=1)
```

Lines 57-59 giảm tỉ lệ học của bộ tối ưu hóa của chúng ta và sau đó biên dịch lại mô hình.

Tiếp theo, chúng ta điều chỉnh lại mô hình trên các ví dụ bất lợi (adversarial examples) (Dòng 63-66).

Cuối cùng, chúng ta sẽ thực hiện một lượt đánh giá cuối cùng:

 → [Launch Jupyter Notebook on Google Colab](#)

 → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
68. # now that our model is fine-tuned we should evaluate it on the test
69. # set (i.e., non-adversarial) again to see if performance has degraded
70. (loss, acc) = model.evaluate(x=testX, y=testY, verbose=0)
71. print("")
72. print("[INFO] normal testing images *after* fine-tuning:")
73. print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
74.
75. # do a final evaluation of the model on the adversarial images
76. (loss, acc) = model.evaluate(x=advX, y=advY, verbose=0)
77. print("[INFO] adversarial images *after* fine-tuning:")
78. print("[INFO] loss: {:.4f}, acc: {:.4f}".format(loss, acc))
```

Chúng ta cần đánh giá lại độ chính xác của mô hình trên cả tập kiểm tra ban đầu (Lines 70-73) và các ví dụ tấn công phía sau (Lines 76-78) sau khi điều chỉnh lại.

Như chúng ta sẽ thấy trong phần tiếp theo, việc điều chỉnh lại mạng CNN trên những ví dụ phức tạp này giúp cho mô hình của chúng ta đưa ra dự đoán chính xác cho cả ảnh gốc và ảnh được tạo ra bởi kỹ thuật tấn công adversarial.

Adversarial image defense results

Bây giờ chúng ta đã sẵn sàng để huấn luyện CNN để bảo vệ chống lại các cuộc tấn công hình ảnh gian lận!

Bắt đầu bằng cách truy cập phần "Tải xuống" của hướng dẫn này để lấy mã nguồn. Từ đó, mở terminal và thực thi lệnh sau:

CO → [Launch Jupyter Notebook on Google Colab](#)

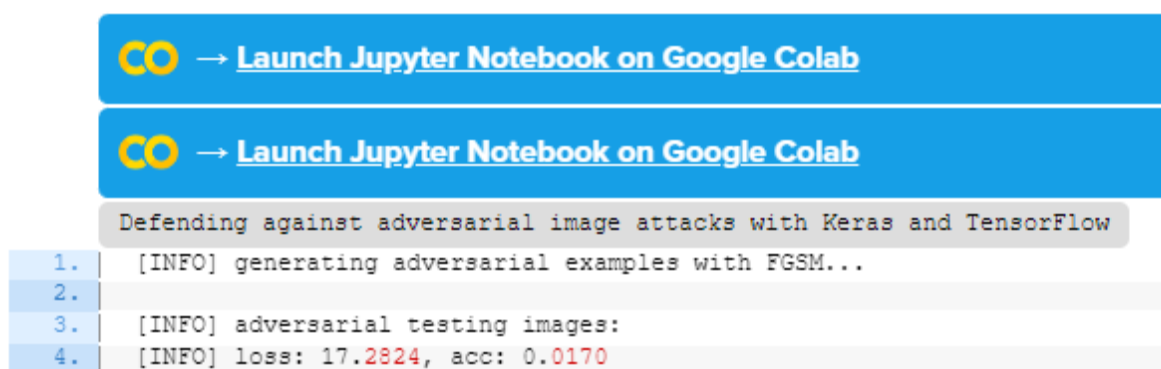
CO → [Launch Jupyter Notebook on Google Colab](#)

Defending against adversarial image attacks with Keras and TensorFlow

```
1. $ time python train_adversarial_defense.py
2. [INFO] loading MNIST dataset...
3. [INFO] compiling model...
4. [INFO] training network...
5. Epoch 1/20
6. 938/938 [=====] - 12s 13ms/step - loss: 0.1973
   - accuracy: 0.9402 - val_loss: 0.0589 - val_accuracy: 0.9809
7. Epoch 2/20
8. 938/938 [=====] - 12s 12ms/step - loss: 0.0781
   - accuracy: 0.9762 - val_loss: 0.0453 - val_accuracy: 0.9838
9. Epoch 3/20
10. 938/938 [=====] - 12s 13ms/step - loss: 0.0599
    - accuracy: 0.9814 - val_loss: 0.0410 - val_accuracy: 0.9868
11. ...
12. Epoch 18/20
13. 938/938 [=====] - 11s 12ms/step - loss: 0.0103
    - accuracy: 0.9963 - val_loss: 0.0476 - val_accuracy: 0.9883
14. Epoch 19/20
15. 938/938 [=====] - 11s 12ms/step - loss: 0.0091
    - accuracy: 0.9967 - val_loss: 0.0420 - val_accuracy: 0.9889
16. Epoch 20/20
17. 938/938 [=====] - 11s 12ms/step - loss: 0.0087
    - accuracy: 0.9970 - val_loss: 0.0443 - val_accuracy: 0.9892
18. [INFO] normal testing images:
19. [INFO] loss: 0.0443, acc: 0.9892
```

Ở đây, bạn có thể thấy chúng tôi đã huấn luyện CNN của mình trên tập dữ liệu MNIST trong 20 epochs. Chúng tôi đã đạt được độ chính xác 99,70% trên tập huấn luyện và 98,92% độ chính xác trên tập kiểm tra của chúng tôi, cho thấy rằng CNN của chúng tôi đang làm rất tốt việc dự đoán các số.

Tuy nhiên, mô hình "cao độ chính xác" này là quá kém và không chính xác khi chúng ta tạo ra một tập hợp 10.000 hình ảnh phản công và yêu cầu CNN phân loại chúng:



```
CO → Launch Jupyter Notebook on Google Colab
CO → Launch Jupyter Notebook on Google Colab
Defending against adversarial image attacks with Keras and TensorFlow
1. [INFO] generating adversarial examples with FGSM...
2.
3. [INFO] adversarial testing images:
4. [INFO] loss: 17.2824, acc: 0.0170
```

Như bạn có thể thấy, độ chính xác của chúng ta giảm từ 98,92% ban đầu xuống còn 1,7%.

Rõ ràng, CNN của chúng ta đã thất bại hoàn toàn trên những hình ảnh đối kháng.

Tuy nhiên, hy vọng vẫn còn! Bây giờ chúng ta sẽ tinh chỉnh lại CNN của mình trên tập hình ảnh đối kháng 10.000:

```

1. [INFO] re-compiling model...
2. [INFO] fine-tuning network on adversarial examples...
3. Epoch 1/10
4. 157/157 [=====] - 2s 12ms/step - loss: 8.0170 -
   accuracy: 0.2455
5. Epoch 2/10
6. 157/157 [=====] - 2s 11ms/step - loss: 1.9634 -
   accuracy: 0.7082
7. Epoch 3/10
8. 157/157 [=====] - 2s 11ms/step - loss: 0.7707 -
   accuracy: 0.8612
9. ...
10. Epoch 8/10
11. 157/157 [=====] - 2s 11ms/step - loss: 0.1186 -
    accuracy: 0.9701
12. Epoch 9/10
13. 157/157 [=====] - 2s 12ms/step - loss: 0.0894 -
    accuracy: 0.9780
14. Epoch 10/10
15. 157/157 [=====] - 2s 12ms/step - loss: 0.0717 -
    accuracy: 0.9817

```

Bây giờ chúng ta đang đạt được độ chính xác khoảng 98% trên các hình ảnh gây ảnh hưởng sau khi được điều chỉnh.

Hãy quay trở lại và đánh giá lại CNN trên cả bộ kiểm tra ban đầu và hình ảnh phản công của chúng ta:

```

Defending against adversarial image attacks with Keras and TensorFlow
1. [INFO] normal testing images *after* fine-tuning:
2. [INFO] loss: 0.0594, acc: 0.9844
3.
4. [INFO] adversarial images *after* fine-tuning:
5. [INFO] loss: 0.0366, acc: 0.9906
6.
7. real    5m12.753s
8. user    12m42.125s
9. sys     10m0.498s

```

Ban đầu, mô hình CNN của chúng ta đạt độ chính xác 98,92% trên bộ kiểm tra của chúng ta.

Độ chính xác đã giảm xuống trên bộ kiểm tra khoảng 0,5%, nhưng tin vui là chúng ta đang đạt được độ chính xác 99% khi phân loại các hình ảnh đối thủ, cho thấy rằng:

Mô hình của chúng ta có thể đưa ra dự đoán chính xác trên các hình ảnh ban đầu không bị thay đổi từ tập dữ liệu MNIST.

Chúng ta cũng có thể đưa ra dự đoán chính xác trên các hình ảnh đối thủ được tạo ra (tức là chúng ta đã thành công trong việc phòng thủ chống lại chúng).

How else can we defend against adversarial attacks?

fine-tuning một mô hình trên hình ảnh đối nghịch là chỉ một trong số các cách để phòng chống các cuộc tấn công đối nghịch.

Một cách tốt hơn là kết hợp hình ảnh đối nghịch với các hình ảnh gốc trong quá trình huấn luyện.

Kết quả là một mô hình mạnh mẽ hơn có khả năng phòng chống các cuộc tấn công đối nghịch vì mô hình tạo ra các hình ảnh đối nghịch riêng của mình trong mỗi lô, do đó liên tục cải thiện chính nó thay vì phải dựa trên một vòng đơn giản của việc fine-tuning sau khi huấn luyện.

Chúng ta sẽ đề cập đến phương pháp "mixed batch adversarial training" này trong bài hướng dẫn tiếp theo vào tuần tới.

Credits and references

The FGSM and data generator implementation were inspired by Sebastian Theiler's excellent article on adversarial attacks and defenses. A huge shoutout and thank you to Sebastian for sharing his knowledge.

Summary

Trong bài hướng dẫn này, bạn đã học cách phòng thủ trước các cuộc tấn công hình ảnh đối kháng bằng cách sử dụng Keras và TensorFlow.

Phương pháp phòng thủ ảnh đối kháng của chúng ta hoạt động bằng cách:

Huấn luyện một CNN trên bộ dữ liệu của chúng ta

Tạo ra một tập hợp các hình ảnh đối kháng bằng cách sử dụng mô hình đã huấn luyện

Tinh chỉnh lại mô hình của chúng ta trên các hình ảnh đối kháng

Kết quả là một mô hình vừa:

Chính xác trên các hình ảnh kiểm tra ban đầu

Có khả năng phân loại chính xác các hình ảnh đối kháng

Phương pháp tinh chỉnh để phòng thủ ảnh đối kháng là phương pháp phòng thủ đối kháng cơ bản nhất. Tuần tới bạn sẽ học phương pháp nâng cao hơn mà kết hợp các hình ảnh đối kháng trong các lô dữ liệu được tạo ra liên tục, cho phép mô hình học từ các ví dụ đối kháng "đánh lừa" nó trong mỗi epoch.