train_adversarial_attack

May 16, 2022

[1]:
```
# Import dependencies
from utils.simplecnn import SimpleCNN
from utils.fgsm import generate_image_adversary
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import cifar10
from utils.cifar_10 import load_data
import numpy as np
from PIL import Image, ImageOps, ImageDraw, ImageFont
from copy import deepcopy
import warnings
warnings.filterwarnings("ignore")
```

[ ]:
```
# If facing certificate error while trying to download dataset, try this
# ssl._create_default_https_context = ssl._create_unverified_context
```

[ ]:
```
# Alternatively, extract the dataset downloaded from https://www.cs.toronto.edu/
 ↪~kriz/cifar-10-python.tar.gz
# !tar -zxvf cifar-10-python.tar.gz
```

[2]:
```
# Specify font to draw text on image
font = ImageFont.truetype("utils/arial.ttf", 9)
```

[ ]:
```
# Load the CIFAR-10 dataset. If server is down (error 503), follow the big␣
 ↪comment below.
# If facing certificate error, follow this: https://stackoverflow.com/questions/
 ↪69687794/unable-to-manually-load-cifar10-dataset

# Download/Get the Python3-compatible CIFAR-10 dataset from https://www.cs.
 ↪toronto.edu/~kriz/cifar-10-python.tar.gz or anywhere else.
# Make sure tar.gz file is fully unzipped and in the same location as this .py␣
 ↪file.
# Use "tar -zxvf cifar-10-python.tar.gz" command to completely unzip the␣
 ↪CIFAR-10 dataset to get a directory
# named "cifar-10-batches-py" in the same location as this current .py file.

# print("[INFO] Loading CIFAR-10 dataset...")
```

```
# (trainX, trainY), (testX, testY) = cifar10.load_data()
```

[3]:
```
# Load the CIFAR-10 dataset
print("[INFO] Loading CIFAR-10 dataset...")
(trainX, trainY), (testX, testY) = load_data()
```

```
[INFO] Loading CIFAR-10 dataset…
```

[4]:
```
# Scale the pixel values to the range [0, 1]
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
```

[5]:
```
# Add a channel dimension to the images
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)
```

[6]:
```
# One-hot encode the labels
trainY = to_categorical(trainY, 10)
testY = to_categorical(testY, 10)
```

[7]:
```
# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog",␣
 ↪"horse", "ship", "truck"]
```

[8]:
```
# Initialize the optimizer and the model
print("[INFO] Compiling the model...")
opt = Adam(lr=1e-3)
model = SimpleCNN.build(width=32, height=32, depth=3, classes=10)
model.compile(loss="categorical_crossentropy", optimizer=opt,␣
 ↪metrics=["accuracy"])
```

```
[INFO] Compiling the model…

2022-05-09 15:00:17.483039: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:17.530119: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:17.530556: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:17.532986: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
```

```
node, so returning NUMA node zero
2022-05-09 15:00:17.533385: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:17.533833: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:18.363303: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:18.363618: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:18.363955: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1052] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-05-09 15:00:18.364140: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 15384 MB memory:  -> device:
0, name: Quadro P5000, pci bus id: 0000:00:06.0, compute capability: 6.1
```

[9]:
```
# Train the simple CNN on the CIFAR-10 dataset
print("[INFO] Training the network...")
model.fit(trainX, trainY, validation_data=(testX, testY), batch_size=64,␣
  ↪epochs=10, verbose=1)
```

```
[INFO] Training the network…
Epoch 1/10

2022-05-09 15:00:22.010018: I tensorflow/stream_executor/cuda/cuda_dnn.cc:377]
Loaded cuDNN version 8302

782/782 [==============================] - 7s 6ms/step - loss: 1.6461 -
accuracy: 0.4433 - val_loss: 1.3289 - val_accuracy: 0.5276
Epoch 2/10
782/782 [==============================] - 5s 6ms/step - loss: 1.1675 -
accuracy: 0.5864 - val_loss: 1.1675 - val_accuracy: 0.5832
Epoch 3/10
782/782 [==============================] - 4s 5ms/step - loss: 1.0105 -
accuracy: 0.6452 - val_loss: 1.0031 - val_accuracy: 0.6385
Epoch 4/10
782/782 [==============================] - 5s 6ms/step - loss: 0.9030 -
accuracy: 0.6815 - val_loss: 1.0358 - val_accuracy: 0.6362
Epoch 5/10
```

```
782/782 [==============================] - 4s 6ms/step - loss: 0.8248 -
accuracy: 0.7104 - val_loss: 0.9967 - val_accuracy: 0.6520
Epoch 6/10
782/782 [==============================] - 5s 6ms/step - loss: 0.7600 -
accuracy: 0.7330 - val_loss: 0.9998 - val_accuracy: 0.6567
Epoch 7/10
782/782 [==============================] - 5s 6ms/step - loss: 0.7022 -
accuracy: 0.7519 - val_loss: 0.9553 - val_accuracy: 0.6714
Epoch 8/10
782/782 [==============================] - 4s 6ms/step - loss: 0.6536 -
accuracy: 0.7694 - val_loss: 0.9616 - val_accuracy: 0.6756
Epoch 9/10
782/782 [==============================] - 5s 6ms/step - loss: 0.5990 -
accuracy: 0.7874 - val_loss: 1.1003 - val_accuracy: 0.6466
Epoch 10/10
782/782 [==============================] - 4s 6ms/step - loss: 0.5660 -
accuracy: 0.7974 - val_loss: 1.0484 - val_accuracy: 0.6567
```

[9]:
```
<keras.callbacks.History at 0x7ff8fc2c6b80>
```

[ ]:
```
model.save('naive_model')
```

[10]:
```
# Make predictions on the testing set for the model trained on non-adversarial␣
 ↪images
(loss, acc) = model.evaluate(x=testX, y=testY, verbose=0)
print("[INFO] Loss: {:.4f}, Accuracy: {:.4f}".format(loss, acc))
```

```
[INFO] Loss: 1.0484, Accuracy: 0.6567
```

1 FGSM Attack

[11]:
```
# Grab the current image and label
image = testX[27]
label = testY[27]
```

[12]:
```
# Generate an image adversary for the current image and make a prediction on␣
 ↪the adversary image
adversary = generate_image_adversary(model, image.reshape(1, 32, 32, 3), label,␣
 ↪eps=0.01)
pred = model.predict(adversary)
```

[13]:
```
# Scale both the original image and the adversary image to the range [0, 255]
# and convert them to unsigned 8-bit integers
adversary = adversary.reshape((32, 32, 3)) * 255
adversary = np.clip(adversary, 0, 255).astype("uint8")
image = image.reshape((32, 32, 3)) * 255
image = image.astype("uint8")
```

[14]:
```
# Resize the images in order to visualize them later
image = Image.fromarray(image).resize((96, 96))
adversary = Image.fromarray(adversary).resize((96, 96))
```

[15]:
```
# Determine the predicted label for both the original image and the adversarial␣
 ↪image
imagePred = label.argmax()
adversaryPred = pred[0].argmax()
```

[16]:
```
# If the image prediction does not match with the adversarial prediction then␣
 ↪update the color
color = (0, 255, 0)
if imagePred != adversaryPred:
    color = (255, 0, 0)
```

[17]:
```
# Draw the predictions on the respective output images
image_copy = deepcopy(image)
adversary_copy = deepcopy(adversary)
draw_image = ImageDraw.Draw(image_copy)
draw_image.text((10, 10), "{}".format(labelNames[imagePred]), (0,255,0),␣
 ↪font=font)
draw_adversary = ImageDraw.Draw(adversary_copy)
draw_adversary.text((10, 10), "{}".format(labelNames[adversaryPred]), color,␣
 ↪font=font)
```

[18]:
```
# Stack the two images horizontally and then show the original image and its␣
 ↪adversary
output = np.hstack([image_copy, adversary_copy])
Image.fromarray(output)
```

[18]:

[24]:
```
# # Loop over a sample of the testing images
# for i in np.random.choice(np.arange(0, len(testX)), size=(10,)):
#     # Grab the current image and label
#     image = testX[i]
#     label = testY[i]
```

```
#     # Generate an image adversary for the current image and make a prediction␣
 ↪on the adversary image
#     adversary = generate_image_adversary(model, image.reshape(1, 32, 32, 3),␣
 ↪label, eps=0.01)
#     pred = model.predict(adversary)
#     # Scale both the original image and the adversary image to the range [0,␣
 ↪255]
#     # and convert them to unsigned 8-bit integers
#     adversary = adversary.reshape((32, 32, 3)) * 255
#     adversary = np.clip(adversary, 0, 255).astype("uint8")
#     image = image.reshape((32, 32, 3)) * 255
#     image = image.astype("uint8")
#     # Resize the images in order to visualize them later
#     image = Image.fromarray(image).resize((96, 96))
#     adversary = Image.fromarray(adversary).resize((96, 96))
#     # Determine the predicted label for both the original image and the␣
 ↪adversarial image
#     imagePred = label.argmax()
#     adversaryPred = pred[0].argmax()
#     color = (0, 255, 0)
#     # If the image prediction does not match with the adversarial prediction␣
 ↪then update the color
#     if imagePred != adversaryPred:
#         color = (255, 0, 0)
#     # Draw the predictions on the respective output images
#     image_copy = deepcopy(image)
#     adversary_copy = deepcopy(adversary)
#     draw_image = ImageDraw.Draw(image_copy)
#     draw_image.text((10, 10), "{}".format(labelNames[imagePred]), (0,255,0),␣
 ↪font=font)
#     draw_adversary = ImageDraw.Draw(adversary_copy)
#     draw_adversary.text((10, 10), "{}".format(labelNames[adversaryPred]),␣
 ↪color, font=font)
#     # Stack the two images horizontally and then show the original image and␣
 ↪its adversary
#     output = np.hstack([image_copy, adversary_copy])
#     Image.fromarray(output)
```

[4]:
```
# export ipynb as pdf
# !apt-get install -y pandoc
# !apt-get -y update && DEBIAN_FRONTEND=noninteractive apt-get install -y␣
 ↪texlive-xetex texlive-fonts-recommended texlive-plain-generic
```