Fast_Gradient_Sign_Method_Adversarial_Attack

April 22, 2022

1 Fast Gradient Sign Method Adversarial Attack

Code based on University of Amsterdam Deep Learning Course.

1.1 Import Libraries

```python
## Standard libraries
import os
import json
import math
import time
import numpy as np
import scipy.linalg

## Imports for plotting
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg', 'pdf') # For export
from matplotlib.colors import to_rgb
import matplotlib
matplotlib.rcParams['lines.linewidth'] = 2.0
import seaborn as sns
sns.set()

## Progress bar
from tqdm.notebook import tqdm

## PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as data
import torch.optim as optim
# Torchvision
import torchvision
from torchvision.datasets import CIFAR10
from torchvision import transforms
```

```python
# PyTorch Lightning
try:
    import pytorch_lightning as pl
except ModuleNotFoundError: # Google Colab does not have PyTorch Lightning␣
 ↪installed by default. Hence, we do it here if necessary
    !pip install --quiet pytorch-lightning>=1.4
    import pytorch_lightning as pl
from pytorch_lightning.callbacks import LearningRateMonitor, ModelCheckpoint

# Path to the folder where the datasets are/should be downloaded (e.g. MNIST)
DATASET_PATH = "../data"
# Path to the folder where the pretrained models are saved
CHECKPOINT_PATH = "../saved_models/tutorial10"

# Setting the seed
pl.seed_everything(42)

# Ensure that all operations are deterministic on GPU (if used) for␣
 ↪reproducibility
torch.backends.cudnn.determinstic = True
torch.backends.cudnn.benchmark = False

# Fetching the device that will be used throughout this notebook
device = torch.device("cpu") if not torch.cuda.is_available() else torch.
 ↪device("cuda:0")
print("Using device", device)
```

```
Global seed set to 42

Using device cpu
```

1.2 Download Dataset and Patches

```python
[2]: import urllib.request
from urllib.error import HTTPError
import zipfile
# Github URL where the dataset is stored for this tutorial
base_url = "https://raw.githubusercontent.com/phlippe/saved_models/main/
 ↪tutorial10/"
# Files to download
pretrained_files = [(DATASET_PATH, "TinyImageNet.zip"), (CHECKPOINT_PATH,␣
 ↪"patches.zip")]
# Create checkpoint path if it doesn't exist yet
os.makedirs(DATASET_PATH, exist_ok=True)
os.makedirs(CHECKPOINT_PATH, exist_ok=True)

# For each file, check whether it already exists. If not, try downloading it.
```

```
for dir_name, file_name in pretrained_files:
    file_path = os.path.join(dir_name, file_name)
    if not os.path.isfile(file_path):
        file_url = base_url + file_name
        print(f"Downloading {file_url}...")
        try:
            urllib.request.urlretrieve(file_url, file_path)
        except HTTPError as e:
            print("Something went wrong. Please try to download the file from␣
 ↪the GDrive folder, or contact the author with the full output including the␣
 ↪following error:\n", e)
        if file_name.endswith(".zip"):
            print("Unzipping file...")
            with zipfile.ZipFile(file_path, 'r') as zip_ref:
                zip_ref.extractall(file_path.rsplit("/",1)[0])
```

Downloading https://raw.githubusercontent.com/phlippe/saved_models/main/tutorial
10/TinyImageNet.zip…
Unzipping file…
Downloading https://raw.githubusercontent.com/phlippe/saved_models/main/tutorial
10/patches.zip…
Unzipping file…

1.3 Load CNN, Dataset and Dataloader

```
[3]: # Load CNN architecture pretrained on ImageNet
os.environ["TORCH_HOME"] = CHECKPOINT_PATH
pretrained_model = torchvision.models.resnet34(pretrained=True)
pretrained_model = pretrained_model.to(device)

# No gradients needed for the network
pretrained_model.eval()
for p in pretrained_model.parameters():
    p.requires_grad = False
```

Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to
../saved_models/tutorial10/hub/checkpoints/resnet34-b627a593.pth

  0%| | 0.00/83.3M [00:00<?, ?B/s]

```
[4]: # Mean and Std from ImageNet
NORM_MEAN = np.array([0.485, 0.456, 0.406])
NORM_STD = np.array([0.229, 0.224, 0.225])
# No resizing and center crop necessary as images are already preprocessed.
plain_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=NORM_MEAN,
                         std=NORM_STD)
```

```
])

# Load dataset and create data loader
imagenet_path = os.path.join(DATASET_PATH, "TinyImageNet/")
assert os.path.isdir(imagenet_path), f"Could not find the ImageNet dataset at␣
 ↪expected path \"{imagenet_path}\". " + \
                                    f"Please make sure to have downloaded the␣
 ↪ImageNet dataset here, or change the {DATASET_PATH} variable."
dataset = torchvision.datasets.ImageFolder(root=imagenet_path,␣
 ↪transform=plain_transforms)
data_loader = data.DataLoader(dataset, batch_size=32, shuffle=False,␣
 ↪drop_last=False, num_workers=2)

# Load label names to interpret the label numbers 0 to 999
with open(os.path.join(imagenet_path, "label_list.json"), "r") as f:
    label_names = json.load(f)

def get_label_index(lab_str):
    assert lab_str in label_names, f"Label \"{lab_str}\" not found. Check the␣
 ↪spelling of the class."
    return label_names.index(lab_str)
```

1.4 Define Auxiliar Functions

- Show Prediction
- Fast Gradient Sign Method

```
[5]: def show_prediction(img, label, pred, K=5, adv_img=None, noise=None):

         if isinstance(img, torch.Tensor):
             # Tensor image to numpy
             img = img.cpu().permute(1, 2, 0).numpy()
             img = (img * NORM_STD[None,None]) + NORM_MEAN[None,None]
             img = np.clip(img, a_min=0.0, a_max=1.0)
             label = label.item()

         # Plot on the left the image with the true label as title.
         # On the right, have a horizontal bar plot with the top k predictions␣
 ↪including probabilities
         if noise is None or adv_img is None:
             fig, ax = plt.subplots(1, 2, figsize=(10,2),␣
 ↪gridspec_kw={'width_ratios': [1, 1]})
         else:
             fig, ax = plt.subplots(1, 5, figsize=(12,2),␣
 ↪gridspec_kw={'width_ratios': [1, 1, 1, 1, 2]})

         ax[0].imshow(img)
```

```python
        ax[0].set_title(label_names[label])
        ax[0].axis('off')

        if adv_img is not None and noise is not None:
            # Visualize adversarial images
            adv_img = adv_img.cpu().permute(1, 2, 0).numpy()
            adv_img = (adv_img * NORM_STD[None,None]) + NORM_MEAN[None,None]
            adv_img = np.clip(adv_img, a_min=0.0, a_max=1.0)
            ax[1].imshow(adv_img)
            ax[1].set_title('Adversarial')
            ax[1].axis('off')
            # Visualize noise
            noise = noise.cpu().permute(1, 2, 0).numpy()
            noise = noise * 0.5 + 0.5 # Scale between 0 to 1
            ax[2].imshow(noise)
            ax[2].set_title('Noise')
            ax[2].axis('off')
            # buffer
            ax[3].axis('off')

        if abs(pred.sum().item() - 1.0) > 1e-4:
            pred = torch.softmax(pred, dim=-1)
        topk_vals, topk_idx = pred.topk(K, dim=-1)
        topk_vals, topk_idx = topk_vals.cpu().numpy(), topk_idx.cpu().numpy()
        ax[-1].barh(np.arange(K), topk_vals*100.0, align='center', color=["C0" if␣
    ↪topk_idx[i]!=label else "C2" for i in range(K)])
        ax[-1].set_yticks(np.arange(K))
        ax[-1].set_yticklabels([label_names[c] for c in topk_idx])
        ax[-1].invert_yaxis()
        ax[-1].set_xlabel('Confidence')
        ax[-1].set_title('Predictions')

        plt.show()
        plt.close()
```

```python
[6]: def fast_gradient_sign_method(model, imgs, labels, epsilon=0.02):
        # Determine prediction of the model
        inp_imgs = imgs.clone().requires_grad_()
        preds = model(inp_imgs.to(device))
        preds = F.log_softmax(preds, dim=-1)
        # Calculate loss by NLL
        loss = -torch.gather(preds, 1, labels.to(device).unsqueeze(dim=-1))
        loss.sum().backward()
        # Update image to adversarial example as written above
        noise_grad = torch.sign(inp_imgs.grad.to(imgs.device))
        fake_imgs = imgs + epsilon * noise_grad
        fake_imgs.detach_()
```
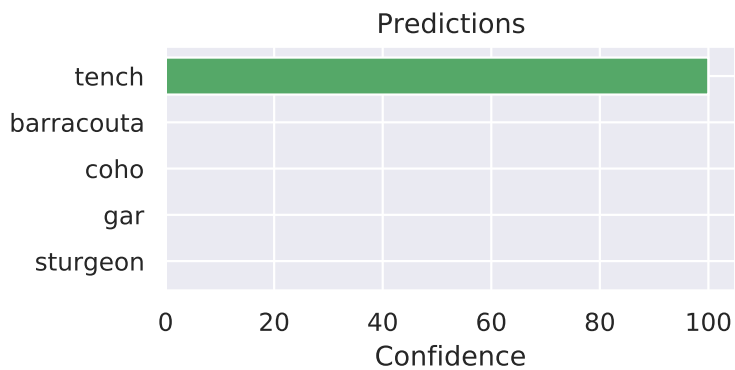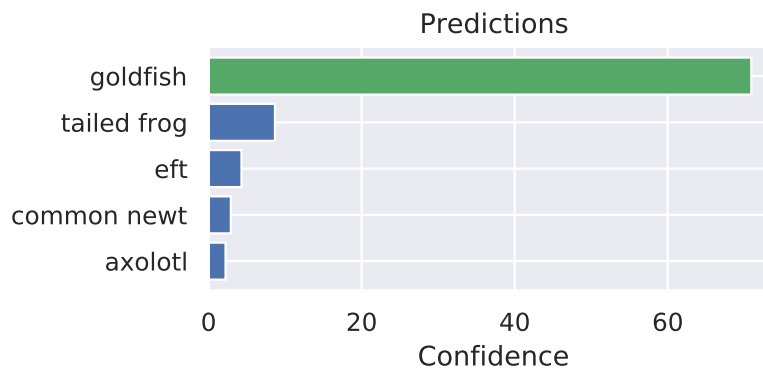
```
        return fake_imgs, noise_grad
```

1.5 Show Correct Predictions

```python
exmp_batch, label_batch = next(iter(data_loader))
with torch.no_grad():
    preds = pretrained_model(exmp_batch.to(device))
for i in range(3,29,5):
    show_prediction(exmp_batch[i], label_batch[i], preds[i])
```
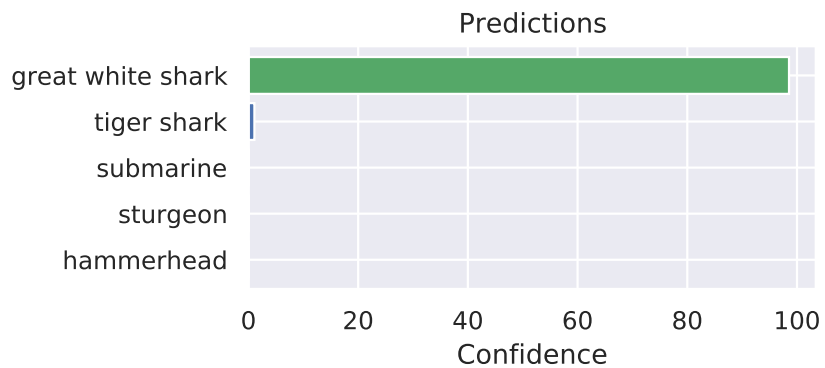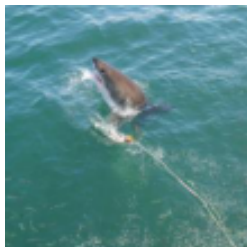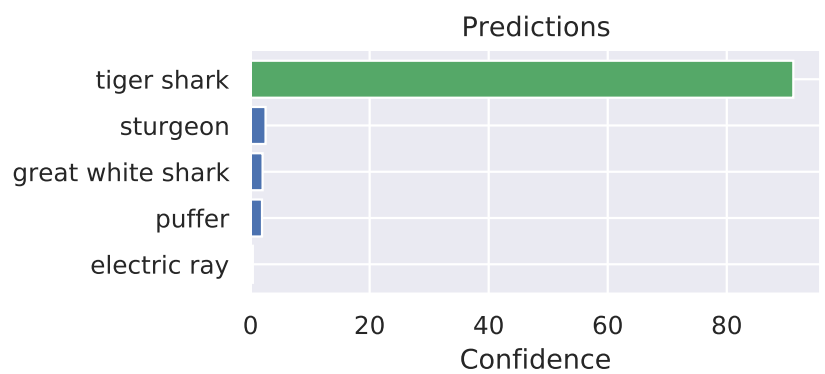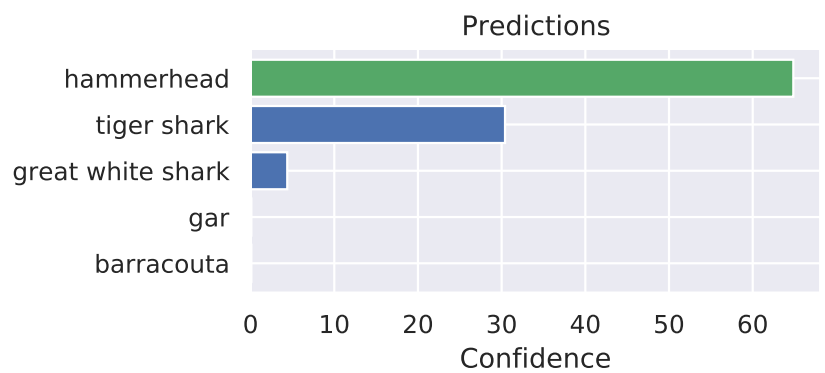
great white shark

Predictions

	Confidence
great white shark	
tiger shark	
submarine	
sturgeon	
hammerhead	

tiger shark

Predictions

	Confidence
tiger shark	
sturgeon	
great white shark	
puffer	
electric ray	

hammerhead

Predictions

	Confidence
hammerhead	
tiger shark	
great white shark	
gar	
barracouta	

electric ray
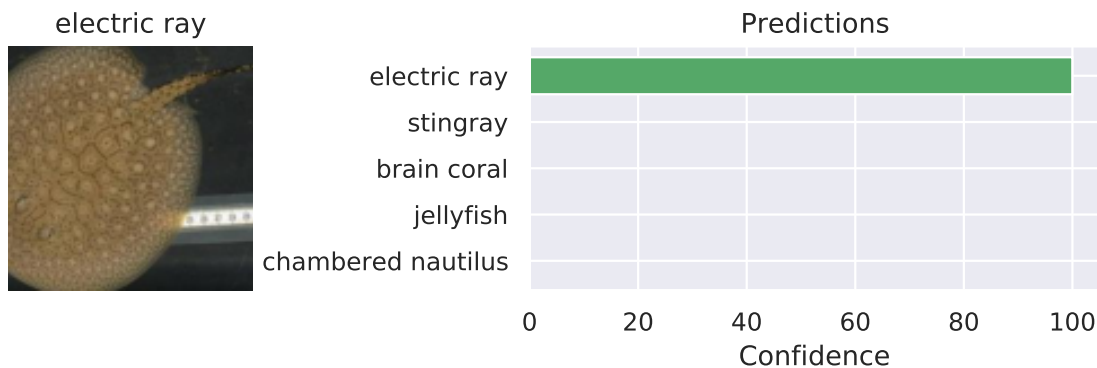
Predictions

1.6 Add Patches and Show Predictions

```
[10]: adv_imgs, noise_grad = fast_gradient_sign_method(pretrained_model, exmp_batch,
      →label_batch, epsilon=0.02)
      with torch.no_grad():
          adv_preds = pretrained_model(adv_imgs.to(device))

      for i in range(2,29,5):
          show_prediction(exmp_batch[i], label_batch[i], adv_preds[i],
      →adv_img=adv_imgs[i], noise=noise_grad[i])
```
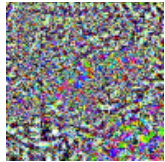

tench   Adversarial   Noise   Predictions

goldfish   Adversarial   Noise   Predictions