OnePixel-атака

А.С. Казимиров

Атаки на нейронные сети

# Перенос предобученной модели на CIFAR10

1. Подключите необходимые библиотеки:

# install dependencies & import libs

!pip3 install torch torchvision

import torch

# Наличие cuda определяется так

DEVICE = 'cuda' if torch.cuda.is\_available() else 'cpu'

print('Using', DEVICE)

1. Загрузите данные и проведите предобработку

import torchvision.datasets as datasets

import torchvision.transforms as transforms

# Выбираем аугментации

train\_trans = transforms.Compose([

    transforms.RandomCrop(32, padding=4),

    transforms.RandomHorizontalFlip(),

    transforms.ToTensor(),

    transforms.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262))

])

test\_trans = transforms.Compose([

    transforms.Resize(32),

    transforms.ToTensor(),

    transforms.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262))

])

# Загружаем данные

dataset = datasets.CIFAR10(root='drive/temp/', download=True, transform=train\_trans)

test\_data = datasets.CIFAR10(root='drive/temp', download=True, train=False, transform=test\_trans)

1. Визуализируйте примеры данных

ind\_2\_name = {0 : 'airplane',

1 : 'automobile',

2 : 'bird',

3 : 'cat',

4 : 'deer',

5 : 'dog',

6 : 'frog',

7 : 'horse',

8 : 'ship',

9 : 'truck'}

import matplotlib.pyplot as plt

import random

def show\_examples(labels, images):

    assert len(labels) == len(images)

    plt.figure(figsize=(20,10))

    for i in range(len(labels)):

        cur\_class = ind\_2\_name[labels[i]]

        plt.subplot(1, 10, i+1)

        plt.title(cur\_class)

        plt.imshow(images[i])

        plt.axis('off')

    plt.show()

indices = [random.randint(0, len(dataset)-1) for i in range(10)]

labels = [dataset.train\_labels[r\_ind] for r\_ind in indices]

images = [dataset.train\_data[r\_ind] for r\_ind in indices]

show\_examples(labels, images)

1. Создайте генераторы батчей

from torch.utils.data import DataLoader

train\_loader = DataLoader(dataset, batch\_size=64, shuffle=True, num\_workers=4)

test\_loader = DataLoader(test\_data, batch\_size=64, num\_workers=4)

inverse\_transform = transforms.Compose([

    transforms.Normalize((-0.491/0.247, -0.482/0.243, -0.447/0.262),

                         (1/0.247, 1/0.243, 1/0.262)),

    transforms.ToPILImage(),

])

images = [

    inverse\_transform(train\_loader.dataset[r\_ind][0]) for r\_ind in indices

]

show\_examples(labels, images)

1. Импортируйте модель squeezenet1\_0 и настройте ее

from torchvision.models import squeezenet1\_0

net = squeezenet1\_0(pretrained=True)

import torch.nn as nn

new\_features = nn.Sequential(\*[nn.Conv2d(3, 96, kernel\_size=3), nn.ReLU()]+list(net.features.children())[3:-2])

net.features = new\_features

net.classifier = nn.Sequential(

    nn.Conv2d(512, 10, kernel\_size=1),

    nn.ReLU(inplace=True),

    nn.AdaptiveAvgPool2d((1,1))

)

net.num\_classes = 10

1. Настройте оптимизатор и дообучите модель

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(nin\_model.parameters(), lr=1e-3)

from tqdm import tqdm

for epoch in range(5):

    reporter = tqdm(enumerate(train\_loader, 0), total=len(train\_loader))

    for i, data in reporter:

        inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

        # Forward pass

        predictions = net(inputs)

        # Estimate predictions quality

        loss = criterion(predictions, targets)

        # Backward pass

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        if i % 50 == 0:

            cur\_loss = loss.item()

            reporter.set\_description('loss %.2f'%cur\_loss)

1. Проверьте качество модели

reporter2 = tqdm(enumerate(test\_loader), total=len(test\_loader))

total\_examples = len(test\_loader.dataset)

num\_matches = 0

for i, data in reporter2:

    with torch.no\_grad():

        inputs, targets = data[0].to(DEVICE), data[1].to(DEVICE)

        predictions = net(inputs)

        \_, label\_predictions = predictions.max(1)

        num\_matches += label\_predictions.eq(targets).sum().item()

print('\nTest accuracy: %.2f%%'%(100.\*num\_matches/total\_examples))

# Onepixel-атака

1. Реализуйте атаку

from random import randint, normalvariate

def init\_population(bounds, popsize):

    population = []

    for \_ in range(0, popsize):

        rand\_x = randint(bounds[0], bounds[1])

        rand\_y = randint(bounds[0], bounds[1])

        rand\_r = normalvariate(0, 1)

        rand\_g = normalvariate(0, 1)

        rand\_b = normalvariate(0, 1)

        population += [(rand\_x, rand\_y,

                        torch.tensor([rand\_r, rand\_g, rand\_b])

                       )]

    return population

def score\_population(population, init\_input, label, model):

    population\_scores = []

    true\_output = torch.tensor([label]).to(DEVICE)

    init\_input = init\_input.to(DEVICE)

    for cell in population:

        cell\_input = init\_input.clone()

        cell\_input[0, :, cell[0], cell[1]] = cell[2]

        cell\_output = model(cell\_input)

        cell\_loss = F.cross\_entropy(cell\_output, true\_output).item()

        population\_scores += [cell\_loss]

    return population\_scores

def recombine\_population(population, F=0.5):

    children\_population = []

    for \_ in range(0, len(population)):

        r\_1 = randint(0, len(population)-1)

        r\_2 = randint(0, len(population)-1)

        r\_3 = randint(0, len(population)-1)

        new\_x = int(population[r\_1][0] + F \* (population[r\_2][0]- population[r\_3][0]))

        new\_x = max(0, min(new\_x, 31))

        new\_y = int(population[r\_1][1] + F \* (population[r\_2][1]- population[r\_3][1]))

        new\_y = max(0, min(new\_y, 31))

        new\_rgb = population[r\_1][2] + F \* (population[r\_2][2] - population[r\_3][2])

        children\_population += [(new\_x, new\_y, new\_rgb)]

    return children\_population

def one\_pixel\_attack(best\_cell, init\_input, label, model):

    # Проводит OnePixel атаку на выбранную картинку и модель

    true\_output = torch.tensor([label]).to(DEVICE)

    init\_input = init\_input.to(DEVICE)

    cell\_input = init\_input.clone()

    cell\_input[0, :, best\_cell[0], best\_cell[1]] = best\_cell[2]

    cell\_output = model(cell\_input)

    cell\_output = F.softmax(cell\_output, dim=1)

    new\_label = cell\_output.max(1)[1].item()

    confidence = cell\_output.max(1)[0].item()

    return cell\_input.cpu(), new\_label, 100.\*confidence

def evolve(population, init\_input, label, model, n\_steps=100, F=0.5):

    old\_population = population

    old\_scores = score\_population(population, init\_input, label, model)

    new\_population = []

    new\_scores = []

    for step in tqdm(range(n\_steps)):

        new\_population = recombine\_population(old\_population, F)

        new\_scores = score\_population(new\_population, init\_input, label, model)

        for i in range(len(new\_population)):

            if new\_scores[i] < old\_scores[i]:

                new\_population[i] = old\_population[i]

        old\_population = new\_population

        old\_scores = new\_scores

    return old\_population, old\_scores

population = init\_population((0, 31), 400)

1. Осуществите атаку и выведите результат

population, scores = evolve(population, init\_tensor, init\_label, net, n\_steps=50)

from numpy import argmax

best\_cell = population[argmax(scores)]

adv\_tensor, adv\_label, conf = one\_pixel\_attack(best\_cell, init\_tensor, init\_label, net)

adv\_class = ind\_2\_name[adv\_label]

adv\_img = get\_img(adv\_tensor)

diff\_img = get\_img((adv\_tensor - init\_tensor) / EPS)

plt.figure(figsize=(15,5))

plt.subplot(131)

plt.title('Init class: ' + init\_class)

plt.imshow(init\_img)

plt.axis('off')

plt.subplot(132)

plt.title('+ one pixel')

plt.imshow(diff\_img)

plt.axis('off')

plt.subplot(133)

plt.title('= Adv class: %s (confidence: %.1f%%)'%(adv\_class, conf))

plt.imshow(adv\_img)

plt.axis('off')

plt.show()