```
# Установка зависимостей
!pip install torch torchvision matplotlib numpy pandas adversarial-
robustness-toolbox tqdm -q

import os
import warnings
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import transforms
from torchvision.datasets import GTSRB  # встроен в torchvision >=
0.12
from torchvision.models import vgg16, resnet50
from tqdm import tqdm  # красивые прогресс-бары
import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")  # убираем предупреждения от ART/PIL
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Используем устройство: {device}")
```

```
Используем устройство: cuda
```

```
data_root = "./gtsrb"
# Трансформации: приводим к 32×32 и нормализуем под ImageNet (т.к.
модели предобучены на нём)
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
])

# Загрузка полных train/test
train_dataset = GTSRB(root=data_root, split='train',
transform=transform, download=True)
test_dataset = GTSRB(root=data_root, split='test',
transform=transform, download=True)

# Подмножества:
attack_1000 = Subset(test_dataset, range(1000))  # первые 1000 — для
нецелевых атак
stop_indices = [i for i, (_, label) in enumerate(test_dataset) if
label == 14][:270]  # класс "Стоп"
stop_subset = Subset(test_dataset, stop_indices)  # 270 изображений —
для целевой атаки

# DataLoader'ы для эффективной загрузки на GPU
batch_size = 64
```

```
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)  # используем test как val
attack_loader = DataLoader(attack_1000, batch_size=batch_size,
shuffle=False)
stop_loader = DataLoader(stop_subset, batch_size=batch_size,
shuffle=False)

print(f"Train: {len(train_dataset)} изображений")
print(f"Test: {len(test_dataset)} изображений")
print(f"Для атак: {len(attack_1000)} изображений")
print(f"Знаков 'Стоп': {len(stop_subset)} изображений")
```

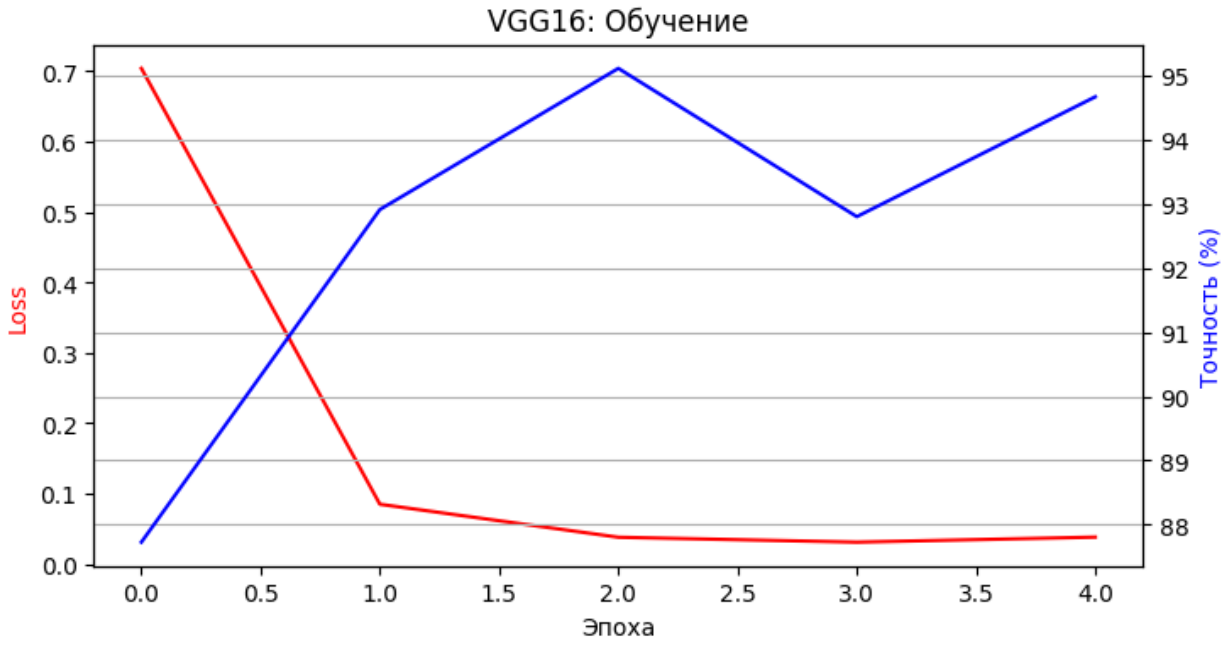
```
100%|██████████| 187M/187M [00:14<00:00, 13.0MB/s]
100%|██████████| 89.0M/89.0M [00:11<00:00, 7.90MB/s]
100%|██████████| 99.6k/99.6k [00:00<00:00, 184kB/s]

Train: 26640 изображений
Test: 12630 изображений
Для атак: 1000 изображений
Знаков 'Стоп': 270 изображений
```

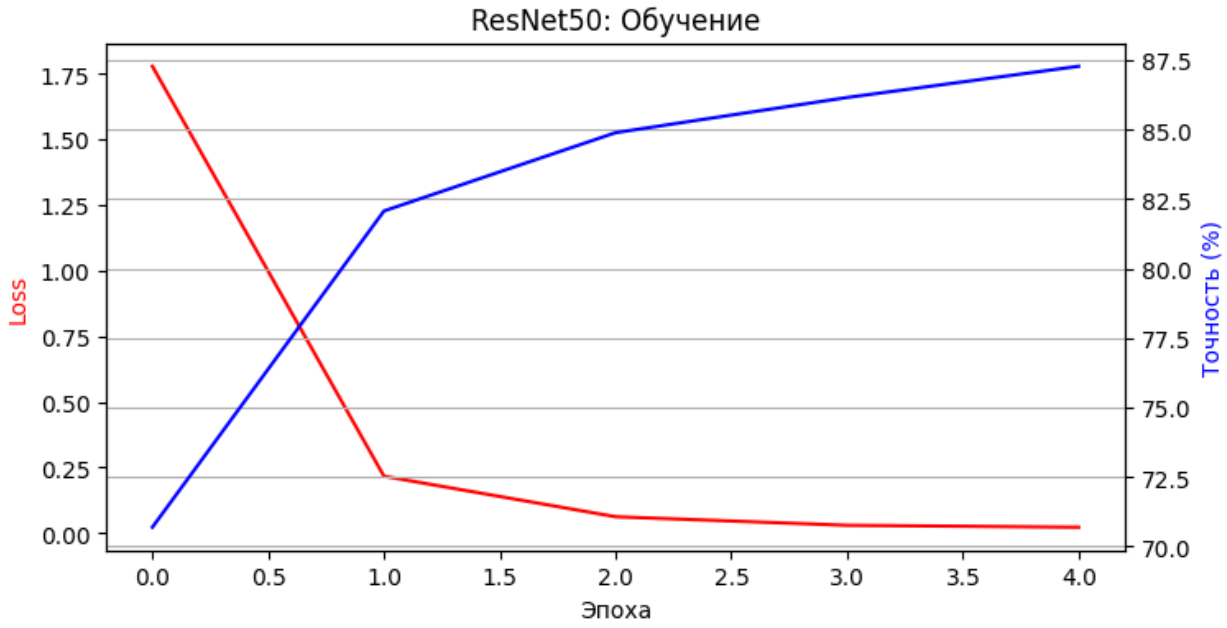
```
def get_model(name, num_classes=43):
    """
    Загружает предобученную модель (ImageNet) и заменяет последний
слой под 43 класса GTSRB.
    """
    if name == "VGG16":
        model = vgg16(weights="IMAGENET1K_V1")
        model.classifier[6] =
nn.Linear(model.classifier[6].in_features, num_classes)
    elif name == "ResNet50":
        model = resnet50(weights="IMAGENET1K_V2")
        model.fc = nn.Linear(model.fc.in_features, num_classes)
    else:
        raise ValueError("Поддерживаются только VGG16 и ResNet50")
    return model.to(device)

def train_model(model_name, epochs=5):
    """
    Обучает модель с Adam-оптимизатором. Возвращает модель, историю
потерь и точности.
    """
    model = get_model(model_name)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)  # можно
подбирать lr
    train_losses, val_accs = [], []
```

```
    for epoch in range(epochs):
        # Обучение
        model.train()
        total_loss = 0
        for x, y in tqdm(train_loader, desc=f"{model_name} Эпоха
{epoch+1}/{epochs}"):
            x, y = x.to(device), y.to(device)
            optimizer.zero_grad()
            loss = criterion(model(x), y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        train_losses.append(total_loss / len(train_loader))

        # Валидация (на тестовом множестве)
        model.eval()
        correct = total = 0
        with torch.no_grad():
            for x, y in val_loader:
                x, y = x.to(device), y.to(device)
                pred = model(x).argmax(1)
                correct += (pred == y).sum().item()
                total += y.size(0)
        val_accs.append(100 * correct / total)
    return model, train_losses, val_accs

def dataloader_to_numpy(loader):
    """
    Конвертирует DataLoader в numpy-массивы (требуется для ART).
    """
    xs, ys = [], []
    for x, y in loader:
        xs.append(x.numpy())
        ys.append(y.numpy())
    return np.concatenate(xs), np.concatenate(ys)

# Обучение VGG16
vgg_model, vgg_loss, vgg_acc = train_model("VGG16", epochs=5)
torch.save(vgg_model.state_dict(), "vgg16_gtsrb.pth")

# Обучение ResNet50
res_model, res_loss, res_acc = train_model("ResNet50", epochs=5)
torch.save(res_model.state_dict(), "resnet50_gtsrb.pth")

# Графики потерь и точности (Рис. 2)
def plot_metrics(loss, acc, title):
    fig, ax1 = plt.subplots(figsize=(8, 4))
    ax1.plot(loss, 'r-', label='Loss')
    ax1.set_xlabel('Эпоха'); ax1.set_ylabel('Loss', color='r')
    ax2 = ax1.twinx()
```

```
    ax2.plot(acc, 'b-', label='Accuracy')
    ax2.set_ylabel('Точность (%)', color='b')
    plt.title(title); plt.grid(True); plt.show()

plot_metrics(vgg_loss, vgg_acc, "VGG16: Обучение")
plot_metrics(res_loss, res_acc, "ResNet50: Обучение")
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth"
to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth

100%|██████████| 528M/528M [00:05<00:00, 110MB/s]
VGG16 Эпоха 1/5: 100%|██████████| 417/417 [00:48<00:00,  8.55it/s]
VGG16 Эпоха 2/5: 100%|██████████| 417/417 [00:44<00:00,  9.47it/s]
VGG16 Эпоха 3/5: 100%|██████████| 417/417 [00:44<00:00,  9.46it/s]
VGG16 Эпоха 4/5: 100%|██████████| 417/417 [00:43<00:00,  9.51it/s]
VGG16 Эпоха 5/5: 100%|██████████| 417/417 [00:43<00:00,  9.54it/s]

Downloading: "https://download.pytorch.org/models/resnet50-
11ad3fa6.pth" to /root/.cache/torch/hub/checkpoints/resnet50-
11ad3fa6.pth

100%|██████████| 97.8M/97.8M [00:00<00:00, 166MB/s]
ResNet50 Эпоха 1/5: 100%|██████████| 417/417 [00:34<00:00, 12.20it/s]
ResNet50 Эпоха 2/5: 100%|██████████| 417/417 [00:33<00:00, 12.46it/s]
ResNet50 Эпоха 3/5: 100%|██████████| 417/417 [00:33<00:00, 12.47it/s]
ResNet50 Эпоха 4/5: 100%|██████████| 417/417 [00:33<00:00, 12.53it/s]
ResNet50 Эпоха 5/5: 100%|██████████| 417/417 [00:33<00:00, 12.43it/s]
```






```
#Конвертирует подмножества в numpy — формат, понятный ART.
x_clean_1000, y_clean_1000 = dataloader_to_numpy(attack_loader)  # для
нецелевых атак
x_stop, y_stop = dataloader_to_numpy(stop_loader)                # для
целевых атак
y_target_30 = np.full_like(y_stop, 1)  # целевой класс: "Ограничение
30" → метка 1

#ART не работает напрямую с torch.nn.Module — нужна обёртка.
from art.estimators.classification import PyTorchClassifier

def make_art_classifier(pytorch_model, nb_classes=43):
    pytorch_model.eval()  # важно: модель должна быть в eval-режиме!
    return PyTorchClassifier(
        model=pytorch_model,
        clip_values=(0, 1),  # значения пикселей в [0,1]
        loss=nn.CrossEntropyLoss(),
        optimizer=optim.Adam(pytorch_model.parameters()),
        input_shape=(3, 32, 32),
        nb_classes=nb_classes,
        device_type="gpu" if torch.cuda.is_available() else "cpu"
    )

art_vgg = make_art_classifier(vgg_model)
art_res = make_art_classifier(res_model)

from art.attacks.evasion import FastGradientMethod,
ProjectedGradientDescent

epsilons = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255,
```

```
50/255, 80/255]

def run_untargeted_attacks(art_model, x, y, eps_list):
    """
    Выполняет нецелевые FGSM и PGD атаки, возвращает точность при
каждом ε.
    """
    results = {"FGSM": [], "PGD": []}
    for eps in eps_list:
        # FGSM: однократный шаг в направлении градиента
        fgsm = FastGradientMethod(art_model, eps=eps)
        acc_fgsm = (art_model.predict(fgsm.generate(x)).argmax(1) ==
y).mean() * 100
        results["FGSM"].append(acc_fgsm)

        # PGD: итеративная версия FGSM с проекцией в L∞-шар
        pgd = ProjectedGradientDescent(art_model, eps=eps,
eps_step=eps/10, max_iter=20)
        acc_pgd = (art_model.predict(pgd.generate(x)).argmax(1) ==
y).mean() * 100
        results["PGD"].append(acc_pgd)

        print(f"ε={eps:.3f} → FGSM: {acc_fgsm:.1f}%, PGD:
{acc_pgd:.1f}%")
    return results

# Запуск атак
vgg_results = run_untargeted_attacks(art_vgg, x_clean_1000,
y_clean_1000, epsilons)
res_results = run_untargeted_attacks(art_res, x_clean_1000,
y_clean_1000, epsilons)
```

{"model_id":"bb510eddae1f4f65a965ad654c9012c8","version_major":2,"vers
ion_minor":0}

```
ε=0.004 → FGSM: 94.7%, PGD: 94.7%
```

{"model_id":"8608fb4afce9489d9dcc7889bfe4531e","version_major":2,"vers
ion_minor":0}

```
ε=0.008 → FGSM: 94.8%, PGD: 94.8%
```

{"model_id":"82d7863499c84bcdb0fcf18b1dd715e1","version_major":2,"vers
ion_minor":0}

```
ε=0.012 → FGSM: 94.8%, PGD: 94.7%
```

{"model_id":"aaab9437e694460aa187053ce80a7340","version_major":2,"vers
ion_minor":0}

```
ε=0.016 → FGSM: 94.6%, PGD: 94.5%
```

{"model_id":"86e350ed56114c519dd36d13b89c8700","version_major":2,"version_minor":0}

ε=0.020 → FGSM: 94.3%, PGD: 94.2%

{"model_id":"6f59b5908ec146a087a6df042e7f8ca5","version_major":2,"version_minor":0}

ε=0.031 → FGSM: 94.8%, PGD: 93.9%

{"model_id":"fb38d5a5af8248548d1252b8863fc7e2","version_major":2,"version_minor":0}

ε=0.039 → FGSM: 94.3%, PGD: 93.8%

{"model_id":"bd32bd6a1d0d4d56ad703b17c6f4db8e","version_major":2,"version_minor":0}

ε=0.078 → FGSM: 92.3%, PGD: 88.2%

{"model_id":"7adef785d50e484d9f07dc84c899a06d","version_major":2,"version_minor":0}

ε=0.196 → FGSM: 82.5%, PGD: 66.9%

{"model_id":"9eda503fae854eabbc9eb9f6fcee3985","version_major":2,"version_minor":0}

ε=0.314 → FGSM: 75.1%, PGD: 52.1%

{"model_id":"6ab7a3b419a240a298b656d3e0983347","version_major":2,"version_minor":0}

ε=0.004 → FGSM: 87.0%, PGD: 86.8%

{"model_id":"e024af5e17274d7c91bc1bffd211c7be","version_major":2,"version_minor":0}

ε=0.008 → FGSM: 86.5%, PGD: 86.3%

{"model_id":"75d84c12a809485bb8f173f42668769d","version_major":2,"version_minor":0}

ε=0.012 → FGSM: 86.4%, PGD: 85.9%

{"model_id":"1114d9e96fcd490faba0951f821c67bd","version_major":2,"version_minor":0}

ε=0.016 → FGSM: 86.2%, PGD: 85.0%

{"model_id":"c0833fba59a0483e8d5e11d296d4ea30","version_major":2,"version_minor":0}

ε=0.020 → FGSM: 85.3%, PGD: 84.5%

{"model_id":"91bdb70aeec24b5597a6d4b996c49e6e","version_major":2,"version_minor":0}

```
ε=0.031 → FGSM: 84.8%, PGD: 82.5%
```

{"model_id":"a9ee431d68e945469f3545608162829a","version_major":2,"version_minor":0}

```
ε=0.039 → FGSM: 83.7%, PGD: 80.9%
```

{"model_id":"6ad5be92f23a4d43b55caa18ee6a3303","version_major":2,"version_minor":0}

```
ε=0.078 → FGSM: 80.4%, PGD: 71.3%
```

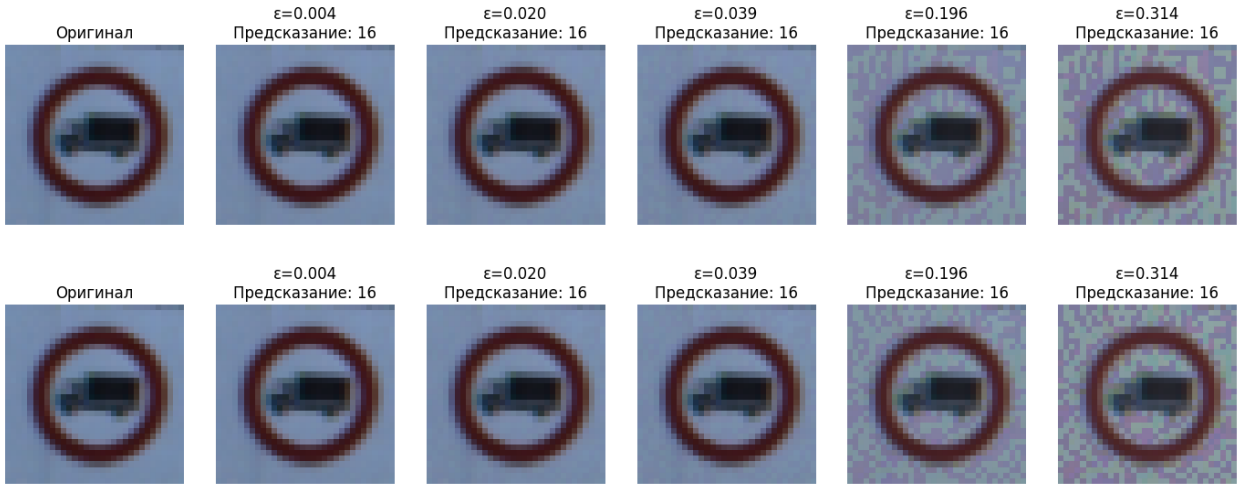
{"model_id":"17a2f5156bda4ee3b24187a6dfde7900","version_major":2,"version_minor":0}

```
ε=0.196 → FGSM: 76.2%, PGD: 52.9%
```

{"model_id":"33fe985ea121449c9d26198102e74de9","version_major":2,"version_minor":0}

```
ε=0.314 → FGSM: 73.8%, PGD: 42.5%
```

```python
# оригинальное и adversarial изображения с предсказаниями.
def denormalize(x):
    """Отменяет нормализацию ImageNet для корректного отображения."""
    mean = np.array([0.485, 0.456, 0.406]).reshape(3,1,1)
    std = np.array([0.229, 0.224, 0.225]).reshape(3,1,1)
    return np.clip(x * std + mean, 0, 1)

def show_fgsm_examples(art_model, x_clean,
eps_list=[1/255,5/255,10/255,50/255,80/255], idx=0):
    fig, ax = plt.subplots(1, len(eps_list)+1, figsize=(14, 2.5))
    ax[0].imshow(denormalize(x_clean[idx]).transpose(1,2,0))
    ax[0].set_title("Оригинал"); ax[0].axis("off")
    for i, eps in enumerate(eps_list):
        fgsm = FastGradientMethod(art_model, eps=eps)
        x_adv = fgsm.generate(x_clean[idx:idx+1])
        pred = art_model.predict(x_adv).argmax()
        ax[i+1].imshow(denormalize(x_adv[0]).transpose(1,2,0))
        ax[i+1].set_title(f"ε={eps:.3f}\nПредсказание: {pred}")
        ax[i+1].axis("off")
    plt.tight_layout(); plt.show()

show_fgsm_examples(art_vgg, x_clean_1000, idx=0)
show_fgsm_examples(art_res, x_clean_1000, idx=0)
```




```
def run_targeted_attacks(art_model, x_clean, y_target, eps_list):
    """
    Целевая атака: модель должна предсказать y_target.
    """
    pgd_rates, fgsm_rates = [], []
    for eps in eps_list:
        # PGD (целевая)
        pgd = ProjectedGradientDescent(art_model, eps=eps,
eps_step=eps/10, max_iter=20, targeted=True)
        x_adv = pgd.generate(x_clean, y=y_target)
        pgd_success = (art_model.predict(x_adv).argmax(1) ==
y_target).mean() * 100
        pgd_rates.append(pgd_success)

        # FGSM (целевая)
        fgsm = FastGradientMethod(art_model, eps=eps, targeted=True)
        x_adv = fgsm.generate(x_clean, y=y_target)
        fgsm_success = (art_model.predict(x_adv).argmax(1) ==
y_target).mean() * 100
        fgsm_rates.append(fgsm_success)

        print(f"ε={eps:.3f} → PGD: {pgd_success:.1f}%, FGSM:
{fgsm_success:.1f}%")
    return pgd_rates, fgsm_rates

eps_target = [1/255, 3/255, 5/255, 10/255, 20/255, 50/255, 80/255]
pgd_vgg, fgsm_vgg = run_targeted_attacks(art_vgg, x_stop, y_target_30,
eps_target)
pgd_res, fgsm_res = run_targeted_attacks(art_res, x_stop, y_target_30,
eps_target)

{"model_id":"87d91488733c433e8dba55f0ab31fe1c","version_major":2,"vers
ion_minor":0}

ε=0.004 → PGD: 0.7%, FGSM: 0.7%
```

{"model_id":"8ce9e3b33daa468bacdec09629005ba5","version_major":2,"version_minor":0}

ε=0.012 → PGD: 0.7%, FGSM: 0.7%

{"model_id":"b853364534004743ac500151da2e7dee","version_major":2,"version_minor":0}

ε=0.020 → PGD: 0.7%, FGSM: 0.7%

{"model_id":"9c043d7df5ee4501baf03fe4589e2f90","version_major":2,"version_minor":0}

ε=0.039 → PGD: 0.7%, FGSM: 0.7%

{"model_id":"3141efa35b0f4a758a296a01e0046403","version_major":2,"version_minor":0}

ε=0.078 → PGD: 2.2%, FGSM: 1.5%

{"model_id":"fea28dab2c9c49358b9f828232a9e560","version_major":2,"version_minor":0}

ε=0.196 → PGD: 28.9%, FGSM: 3.3%

{"model_id":"50465b4b3dcd464fa689307955376a47","version_major":2,"version_minor":0}

ε=0.314 → PGD: 53.0%, FGSM: 5.6%

{"model_id":"0bfc0d7bf706401a8ed4c9733bf7bc11","version_major":2,"version_minor":0}

ε=0.004 → PGD: 0.4%, FGSM: 0.4%

{"model_id":"35ecef78505847d98f2fd4e311e3ede0","version_major":2,"version_minor":0}

ε=0.012 → PGD: 0.7%, FGSM: 0.4%

{"model_id":"13a8d8497ae54b4cb5b988811eff2bbc","version_major":2,"version_minor":0}

ε=0.020 → PGD: 0.7%, FGSM: 0.7%

{"model_id":"c5531fedaab343c280bebee36db0a569","version_major":2,"version_minor":0}

ε=0.039 → PGD: 2.2%, FGSM: 1.1%

{"model_id":"32019d8b9b0440aebe79146b56020588","version_major":2,"version_minor":0}

ε=0.078 → PGD: 5.9%, FGSM: 1.1%

{"model_id":"85a8385b8dc84627930c386c6cfe72dc","version_major":2,"version_minor":0}

```
ε=0.196 → PGD: 25.2%, FGSM: 0.7%
```

{"model_id":"52c3c8e6ef47484696445ad248f132e1","version_major":2,"version_minor":0}

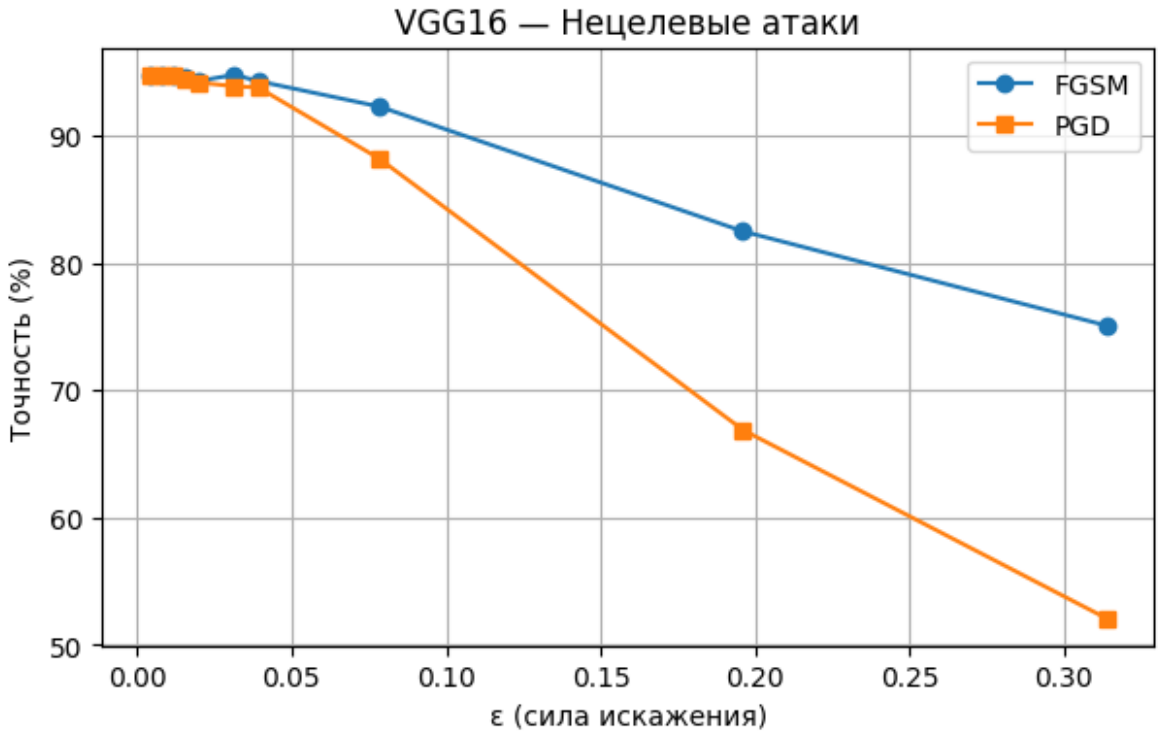
```
ε=0.314 → PGD: 44.1%, FGSM: 1.5%
```

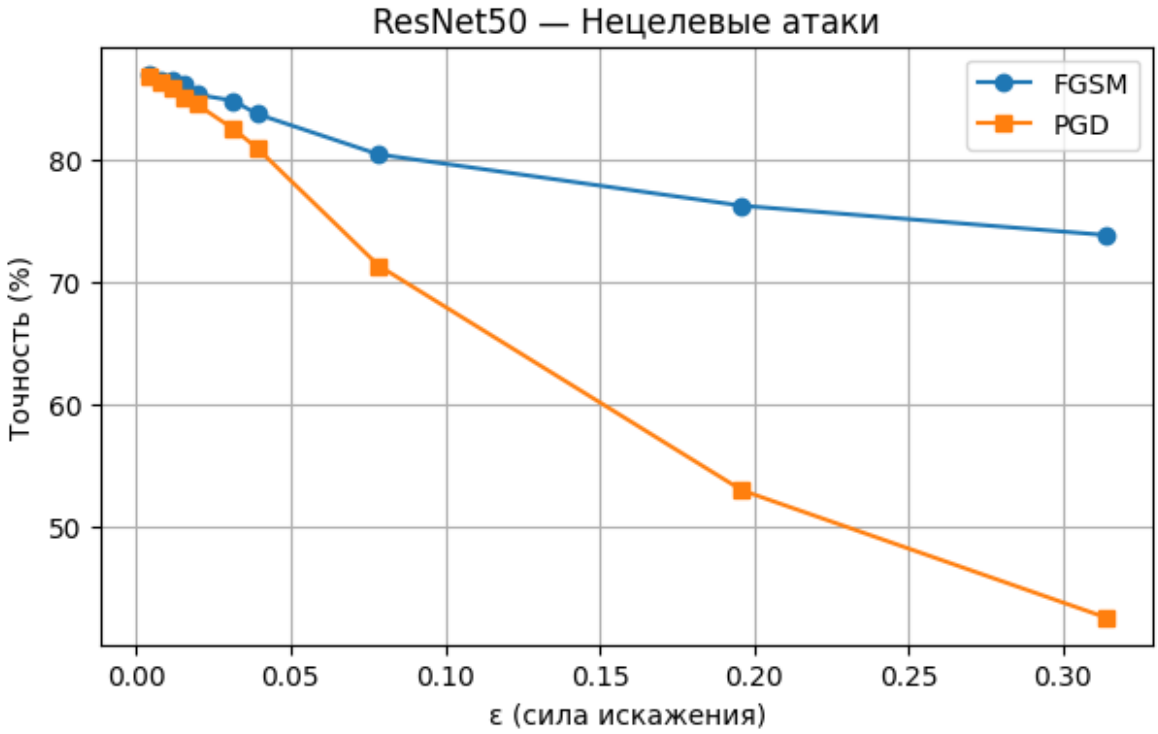
```
# Визуализация целевых атак 5 пар "Стоп → Speed 30".
def show_targeted_examples(art_model, x_clean, y_target, eps=10/255, n=5):
    pgd = ProjectedGradientDescent(art_model, eps=eps, eps_step=eps/10, max_iter=20, targeted=True)
    x_adv = pgd.generate(x_clean[:n], y=y_target[:n])
    fig, ax = plt.subplots(2, n, figsize=(12, 4))
    for i in range(n):
        ax[0,i].imshow(denormalize(x_clean[i]).transpose(1,2,0))
        ax[0,i].set_title("Знак 'Стоп'"); ax[0,i].axis("off")
        pred = art_model.predict(x_adv[i:i+1]).argmax()
        ax[1,i].imshow(denormalize(x_adv[i]).transpose(1,2,0))
        ax[1,i].set_title(f"Предсказано: {pred}"); ax[1,i].axis("off")
    plt.tight_layout(); plt.show()

show_targeted_examples(art_vgg, x_stop, y_target_30, eps=10/255)
```

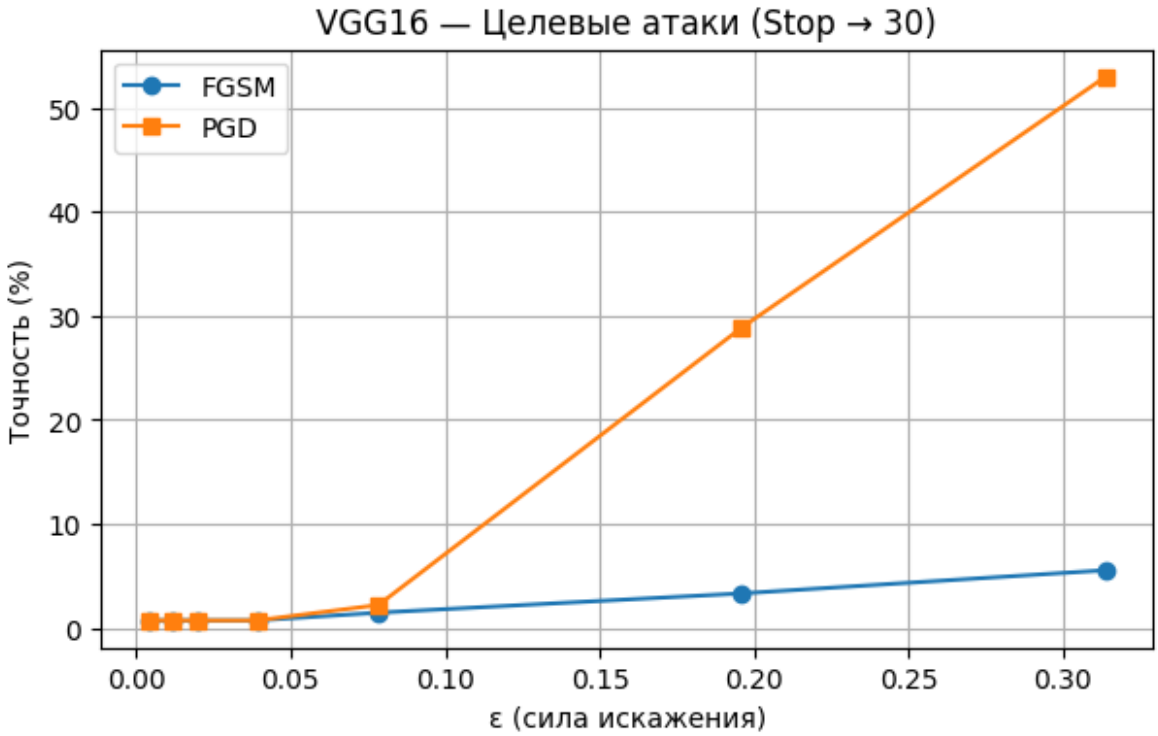
{"model_id":"2a86e1660d8740ed8364591051d4afe3","version_major":2,"version_minor":0}

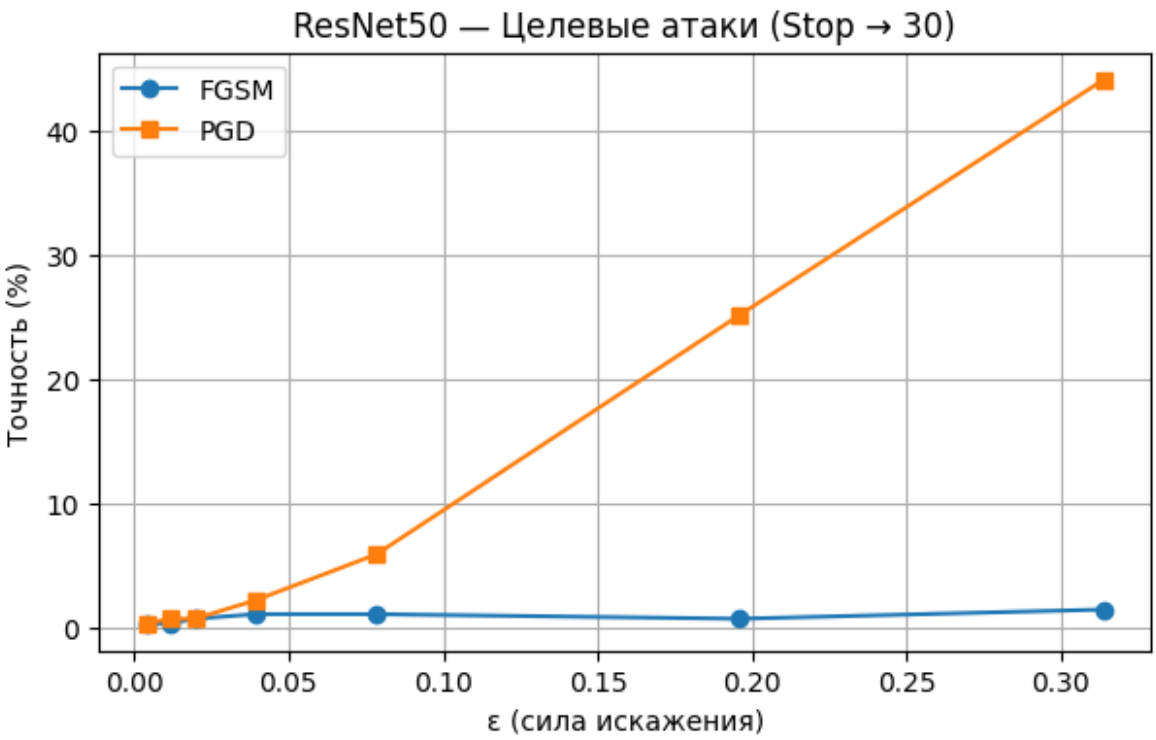

```
# Графики зависимости точности от ε.
def plot_accuracy_curves(eps, fgsm, pgd, title):
    plt.figure(figsize=(7,4))
    plt.plot(eps, fgsm, 'o-', label='FGSM')
    plt.plot(eps, pgd, 's-', label='PGD')
    plt.xlabel('ε (сила искажения)')
```

```
    plt.ylabel('Точность (%)')
    plt.title(title)
    plt.legend(); plt.grid(True)
    plt.show()

# Нецелевые
plot_accuracy_curves(epsilons, vgg_results["FGSM"],
vgg_results["PGD"], "VGG16 — Нецелевые атаки")
plot_accuracy_curves(epsilons, res_results["FGSM"],
res_results["PGD"], "ResNet50 — Нецелевые атаки")

# Целевые
plot_accuracy_curves(eps_target, fgsm_vgg, pgd_vgg, "VGG16 — Целевые
атаки (Stop → 30)")
plot_accuracy_curves(eps_target, fgsm_res, pgd_res, "ResNet50 —
Целевые атаки (Stop → 30)")
```




ResNet50 — Нецелевые атаки
FGSM
PGD
Точность (%)
80
70
60
50
0.00
0.05
0.10
0.15
0.20
0.25
0.30
ε (сила искажения)


VGG16 — Целевые атаки (Stop → 30)
FGSM
PGD
Точность (%)
50
40
30
20
10
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
ε (сила искажения)


ResNet50 — Целевые атаки (Stop → 30)
FGSM
PGD
Точность (%)
0
10
20
30
40
0.00
0.05
0.10
0.15
0.20
0.25
0.30
ε (сила искажения)