Adversarial Attack Project

```python
# Colab Setup Cell — run this first (updated)


# 2. Upload and unzip your TestDataSet.zip
from google.colab import files
import zipfile, io, os
uploaded = files.upload()  # select TestDataSet.zip
zf = zipfile.ZipFile(io.BytesIO(uploaded['TestDataSet.zip']))
zf.extractall('/content/TestDataSet')
print("Unzipped to /content/TestDataSet")

# 3. Remove the macOS metadata folder entirely
!rm -rf /content/TestDataSet/__MACOSX

# 4. Verify structure
!find /content/TestDataSet -maxdepth 2 -type d
```

```
<IPython.core.display.HTML object>
```

```
Saving TestDataSet.zip to TestDataSet.zip
Unzipped to /content/TestDataSet
/content/TestDataSet
/content/TestDataSet/TestDataSet
/content/TestDataSet/TestDataSet/n02708093
/content/TestDataSet/TestDataSet/n02807133
/content/TestDataSet/TestDataSet/n03032252
/content/TestDataSet/TestDataSet/n02825657
/content/TestDataSet/TestDataSet/n02840245
/content/TestDataSet/TestDataSet/n02917067
/content/TestDataSet/TestDataSet/n02883205
/content/TestDataSet/TestDataSet/n02916936
/content/TestDataSet/TestDataSet/n02859443
/content/TestDataSet/TestDataSet/n02797295
/content/TestDataSet/TestDataSet/n02980441
/content/TestDataSet/TestDataSet/n02966193
/content/TestDataSet/TestDataSet/n02814533
/content/TestDataSet/TestDataSet/n02687172
/content/TestDataSet/TestDataSet/n03026506
/content/TestDataSet/TestDataSet/n02978881
/content/TestDataSet/TestDataSet/n02823750
/content/TestDataSet/TestDataSet/n02690373
/content/TestDataSet/TestDataSet/n02799071
/content/TestDataSet/TestDataSet/n02966687
/content/TestDataSet/TestDataSet/n02948072
/content/TestDataSet/TestDataSet/n02676566
/content/TestDataSet/TestDataSet/n02892201
```

```
/content/TestDataSet/TestDataSet/n02672831
/content/TestDataSet/TestDataSet/n03018349
/content/TestDataSet/TestDataSet/n02808440
/content/TestDataSet/TestDataSet/n02776631
/content/TestDataSet/TestDataSet/n02999410
/content/TestDataSet/TestDataSet/n02860847
/content/TestDataSet/TestDataSet/n02791270
/content/TestDataSet/TestDataSet/n02782093
/content/TestDataSet/TestDataSet/n02783161
/content/TestDataSet/TestDataSet/n02894605
/content/TestDataSet/TestDataSet/n03016953
/content/TestDataSet/TestDataSet/n02835271
/content/TestDataSet/TestDataSet/n02871525
/content/TestDataSet/TestDataSet/n02814860
/content/TestDataSet/TestDataSet/n02804610
/content/TestDataSet/TestDataSet/n03017168
/content/TestDataSet/TestDataSet/n02870880
/content/TestDataSet/TestDataSet/n02971356
/content/TestDataSet/TestDataSet/n02909870
/content/TestDataSet/TestDataSet/n02981792
/content/TestDataSet/TestDataSet/n02808304
/content/TestDataSet/TestDataSet/n02786058
/content/TestDataSet/TestDataSet/n02950826
/content/TestDataSet/TestDataSet/n02951358
/content/TestDataSet/TestDataSet/n02841315
/content/TestDataSet/TestDataSet/n02910353
/content/TestDataSet/TestDataSet/n02793495
/content/TestDataSet/TestDataSet/n02834397
/content/TestDataSet/TestDataSet/n02787622
/content/TestDataSet/TestDataSet/n02939185
/content/TestDataSet/TestDataSet/n02837789
/content/TestDataSet/TestDataSet/n03000134
/content/TestDataSet/TestDataSet/n02701002
/content/TestDataSet/TestDataSet/n02843684
/content/TestDataSet/TestDataSet/n03028079
/content/TestDataSet/TestDataSet/n02977058
/content/TestDataSet/TestDataSet/n02965783
/content/TestDataSet/TestDataSet/n02747177
/content/TestDataSet/TestDataSet/n02790996
/content/TestDataSet/TestDataSet/n03000684
/content/TestDataSet/TestDataSet/n02895154
/content/TestDataSet/TestDataSet/n02963159
/content/TestDataSet/TestDataSet/n02692877
/content/TestDataSet/TestDataSet/n02979186
/content/TestDataSet/TestDataSet/n02815834
/content/TestDataSet/TestDataSet/n02879718
/content/TestDataSet/TestDataSet/n02795169
/content/TestDataSet/TestDataSet/n02992529
/content/TestDataSet/TestDataSet/n03000247
```

```
/content/TestDataSet/TestDataSet/n02749479
/content/TestDataSet/TestDataSet/n02704792
/content/TestDataSet/TestDataSet/n03042490
/content/TestDataSet/TestDataSet/n02927161
/content/TestDataSet/TestDataSet/n03041632
/content/TestDataSet/TestDataSet/n03014705
/content/TestDataSet/TestDataSet/n02802426
/content/TestDataSet/TestDataSet/n02906734
/content/TestDataSet/TestDataSet/n02794156
/content/TestDataSet/TestDataSet/n02892767
/content/TestDataSet/TestDataSet/n02777292
/content/TestDataSet/TestDataSet/n02699494
/content/TestDataSet/TestDataSet/n02865351
/content/TestDataSet/TestDataSet/n02804414
/content/TestDataSet/TestDataSet/n02791124
/content/TestDataSet/TestDataSet/n02823428
/content/TestDataSet/TestDataSet/n02769748
/content/TestDataSet/TestDataSet/n02730930
/content/TestDataSet/TestDataSet/n02930766
/content/TestDataSet/TestDataSet/n02727426
/content/TestDataSet/TestDataSet/n02951585
/content/TestDataSet/TestDataSet/n02817516
/content/TestDataSet/TestDataSet/n02788148
/content/TestDataSet/TestDataSet/n02974003
/content/TestDataSet/TestDataSet/n02988304
/content/TestDataSet/TestDataSet/n02869837
/content/TestDataSet/TestDataSet/n02992211
/content/TestDataSet/TestDataSet/n02877765
```

#1: Imports & Setup

```python
import os
import random
import json
import numpy as np
import torch
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader, TensorDataset
from pathlib import Path
from tqdm import tqdm
import matplotlib.pyplot as plt
from PIL import Image

# reproducibility
random.seed(42)
```

```python
np.random.seed(42)
torch.manual_seed(42)

# device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)

# normalization params & clamp tensors
data_mean = [0.485, 0.456, 0.406]
data_std  = [0.229, 0.224, 0.225]
_mean = torch.tensor(data_mean, device=device).view(1,3,1,1)
_std  = torch.tensor(data_std,  device=device).view(1,3,1,1)
clamp_min = (0 - _mean) / _std
clamp_max = (1 - _mean) / _std
clamp_min_spatial = clamp_min.squeeze(0)
clamp_max_spatial = clamp_max.squeeze(0)

# denormalizer for plotting
denorm = transforms.Normalize(
    mean=[-m/s for m,s in zip(data_mean, data_std)],
    std =[1/s    for s    in data_std]
)

# load ImageNet class index
idx_path = Path('imagenet_class_index.json')
assert idx_path.exists(), f"Missing {idx_path}"
with open(idx_path) as f:
    imagenet_idx = json.load(f)
class_to_idx = {v[0]:int(k) for k,v in imagenet_idx.items()}
idx_to_name  = {int(k):v[1] for k,v in imagenet_idx.items()}
print('Loaded', len(class_to_idx), 'classes')
```

2: Utilities (model loader, dataloader, eval, save, visualize)

```python
def load_resnet34():
    """
    Load base ResNet34 model.
    """
    m = 
torchvision.models.resnet34(weights='IMAGENET1K_V1').to(device)
    m.eval()
    return m

def get_loader(path, bs=32):
```

```
    tf = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
        transforms.Normalize(data_mean, data_std)
    ])
    ds = ImageFolder(path, transform=tf)
    fmap = {
        idx: class_to_idx.get(wnid.split('_')[0], idx)
        for wnid, idx in ds.class_to_idx.items()
    }
    def collate(batch):
        imgs, lbls = zip(*batch)
        return torch.stack(imgs), torch.tensor([fmap[l] for l in
lbls])
    return ds, DataLoader(ds, batch_size=bs, shuffle=False,
collate_fn=collate)

def evaluate(model, loader, atk=None, eps=None, topk=(1,5)):
    corr, tot = {k:0 for k in topk}, 0
    advs, advlbl = [], []
    for x,y in tqdm(loader, desc="Evaluating"):
        x,y = x.to(device), y.to(device)
        inp = x if atk is None else atk(model, x.clone(), y, eps)
        if atk is not None:
            advs.extend(inp.cpu()); advlbl.extend(y.cpu())
        with torch.no_grad():
            out = model(inp)
        _, pred = out.topk(max(topk),1,True,True)
        pred = pred.t()
        m = pred.eq(y.view(1,-1).expand_as(pred))
        for k in topk:
            corr[k] += m[:k].any(0).sum().item()
        tot += y.size(0)
    return {k:corr[k]/tot for k in topk}, (torch.stack(advs) if advs
else None), advlbl

def save_set(ds, adv, outdir):
    os.makedirs(outdir, exist_ok=True)
    per = len(adv) // len(ds.classes)
    with torch.no_grad():
        for i,img in enumerate(adv):
            cls = ds.classes[i//per]
            d = Path(outdir)/cls; d.mkdir(exist_ok=True, parents=True)
            den = denorm(img).clamp(0,1)
            arr =
(den.permute(1,2,0).cpu().numpy()*255).astype('uint8')
            Image.fromarray(arr).save(d/f'adv_{i:05d}.png')


# ─── after loading your mapping ────────────────────────────
```

```python
imagenet_idx_to_name = {int(k):v[1] for k,v in imagenet_idx.items()}
idx_to_name = imagenet_idx_to_name  # alias for show_examples


def topk_preds(net, x, idx_to_name, k=5):
    net.eval()
    with torch.no_grad():
        logits = net(x.unsqueeze(0).to(device))
        probs  = F.softmax(logits[0], dim=0)
    topk_probs, topk_idx = probs.topk(k)
    labels = [idx_to_name[i.item()] for i in topk_idx]
    return labels, topk_probs.cpu().numpy()

def plot_topk(labels, probs, ax, title="Top-5 Predictions"):
    y = np.arange(len(labels))[::-1]
    ax.barh(y, probs[::-1], align='center')
    ax.set_yticks(y)
    ax.set_yticklabels(labels[::-1])
    ax.set_xlabel("Confidence")
    ax.set_title(title)
    ax.set_xlim(0,1.0)
    for i,p in enumerate(probs[::-1]):
        ax.text(p + 0.02, i, f"{p:.2f}", va='center')

def show_examples(orig, adv, o_lbl, a_lbl, true_lbl):
    """
    Displays:
      [0] Orig image + pred vs true
      [1] Adv image + pred
      [2] Noise (scaled)
      [3] Top-5 confidences for adv
    """
    # detach
    orig_det = orig.detach()
    adv_det  = adv.detach()

    # images
    o_img = denorm(orig_det).permute(1,2,0).clamp(0,1).cpu().numpy()
    a_img = denorm(adv_det).permute(1,2,0).clamp(0,1).cpu().numpy()

    # noise
    noise = adv_det - orig_det
    mn, mx = noise.min(), noise.max()
    noise_vis = ((noise - mn) / (mx - mn + 1e-8))\
                 .permute(1,2,0).cpu().numpy()

    # top-5 on adv
    adv_labels, adv_probs = topk_preds(model, adv_det, idx_to_name,
k=5)
```

```
    # plot
    fig, axes = plt.subplots(1,4, figsize=(16,4))

    axes[0].imshow(o_img)
    axes[0].set_title(f"Orig: {idx_to_name[o_lbl]}\nTrue:
{idx_to_name[int(true_lbl)]}")
    axes[0].axis('off')

    axes[1].imshow(a_img)
    axes[1].set_title(f"Adv: {idx_to_name[a_lbl]}")
    axes[1].axis('off')

    axes[2].imshow(noise_vis)
    axes[2].set_title("Noise (scaled)")
    axes[2].axis('off')

    plot_topk(adv_labels, adv_probs, axes[3], title="Top-5
Predictions")

    plt.tight_layout()
    plt.show()

### Loading resnet34 model ########
model = load_resnet34()
model.eval()

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
```

```
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
```

```
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

3: Task 1 — Baseline evaluation

```
# Cell 3: Task 1 — Baseline evaluation
ds, ldr = get_loader('./TestDataSet/TestDataSet')
```

```python
acc, _, _ = evaluate(model, ldr)
print(f"Clean Top-1: {acc[1]*100:.2f}%, Top-5: {acc[5]*100:.2f}%")
```

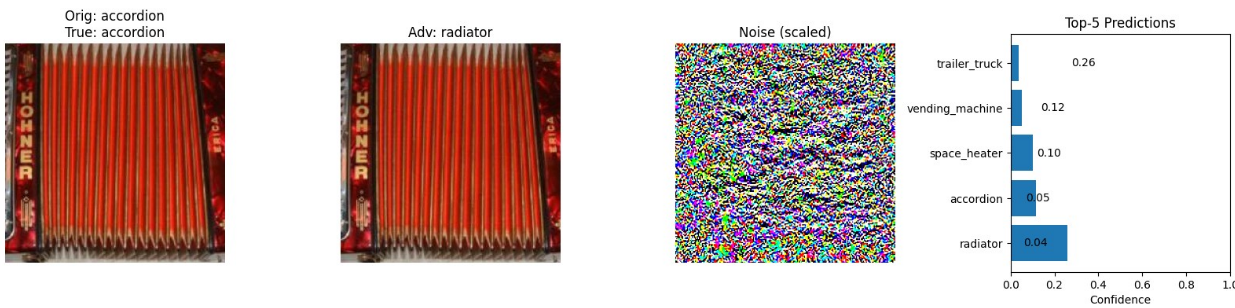
```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100%|██████████| 83.3M/83.3M [00:00<00:00, 174MB/s]
Evaluating: 100%|██████████| 16/16 [00:02<00:00,  5.92it/s]

Clean Top-1: 76.00%, Top-5: 94.20%
```

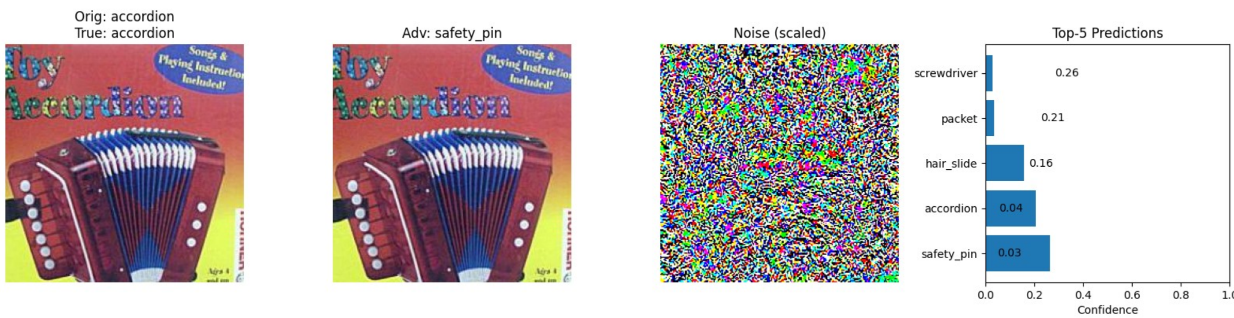
Task 2 — FGSM Attack

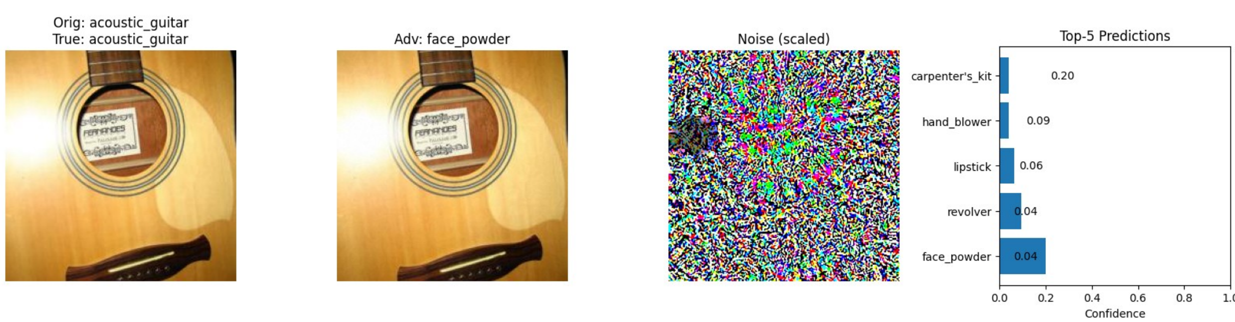
```python
# store baseline Top-1 from Task 1
orig_top1 = acc[1]

def fgsm(model, x, y, eps=0.02):
    x_adv = x.clone().detach().requires_grad_(True)
    loss = F.cross_entropy(model(x_adv), y)
    model.zero_grad()
    loss.backward()
    adv = x_adv + eps * x_adv.grad.sign()
    return adv.clamp(clamp_min, clamp_max)

print("==== Task 2: FGSM (ε=0.02) ====")
acc1, adv1, lb1 = evaluate(model, ldr, fgsm, 0.02)
print(f"FGSM Top-1: {acc1[1]*100:.2f}%, Top-5: {acc1[5]*100:.2f}%")

# verify L∞ constraint
orig_imgs = torch.stack([img for img,_ in ldr.dataset])
max_pert = (adv1 - orig_imgs).abs().view(adv1.size(0), -1).max(1)[0].max()
print(f"Max L∞ perturbation: {max_pert:.6f}")
assert max_pert <= 0.0201, "Exceeded ε = 0.02 bound!"

# check for ≥50% relative drop
drop = (orig_top1 - acc1[1]) / orig_top1
print(f"Relative Top-1 drop: {drop*100:.1f}%")
assert drop >= 0.50, "Relative drop < 50% — consider increasing ε or using stronger method."

# save adversarial set
save_set(ds, adv1, 'AdversarialTestSet1')

# visualize 5 failure cases
count = 0
for i in range(len(adv1)):
```

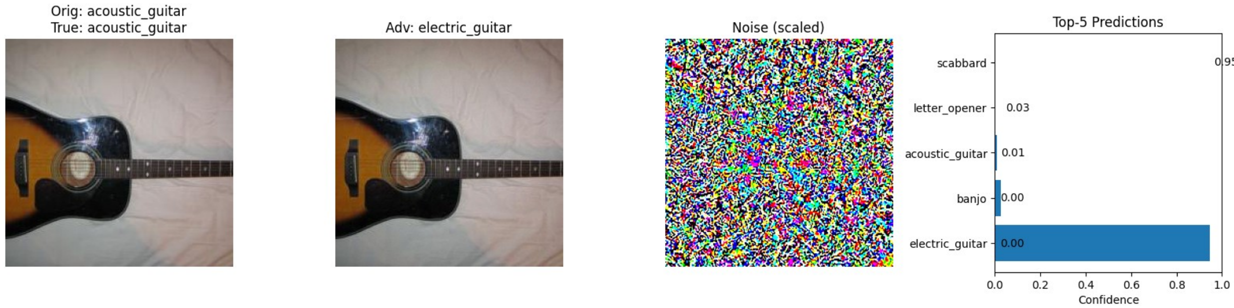
```
    o_lbl = model(orig_imgs[i:i+1].to(device)).argmax(1).item()
    a_lbl = model(adv1[i:i+1].to(device)).argmax(1).item()
    if o_lbl == lb1[i] and a_lbl != lb1[i]:
        show_examples(orig_imgs[i], adv1[i], o_lbl, a_lbl, lb1[i])
        count += 1
        if count >= 5:
            break
```

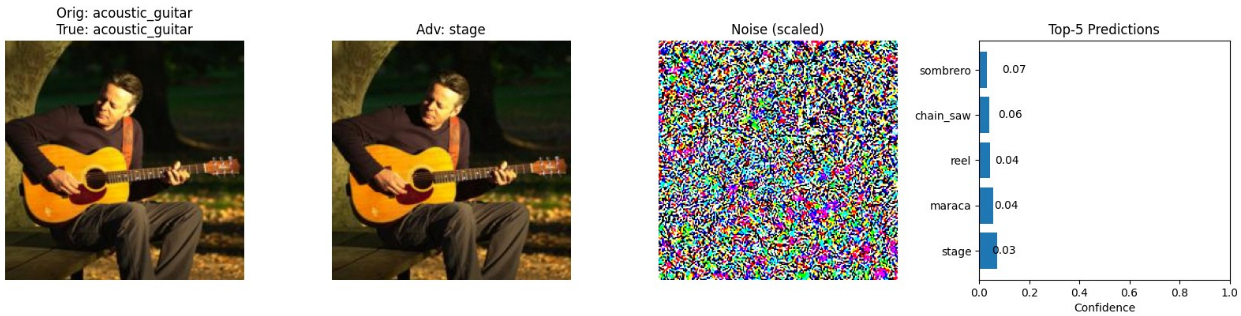
```
==== Task 2: FGSM (ε=0.02) ====

Evaluating: 100%|██████████| 16/16 [00:04<00:00,  3.61it/s]

FGSM Top-1: 6.20%, Top-5: 35.40%
Max L∞ perturbation: 0.020000
Relative Top-1 drop: 91.8%
```











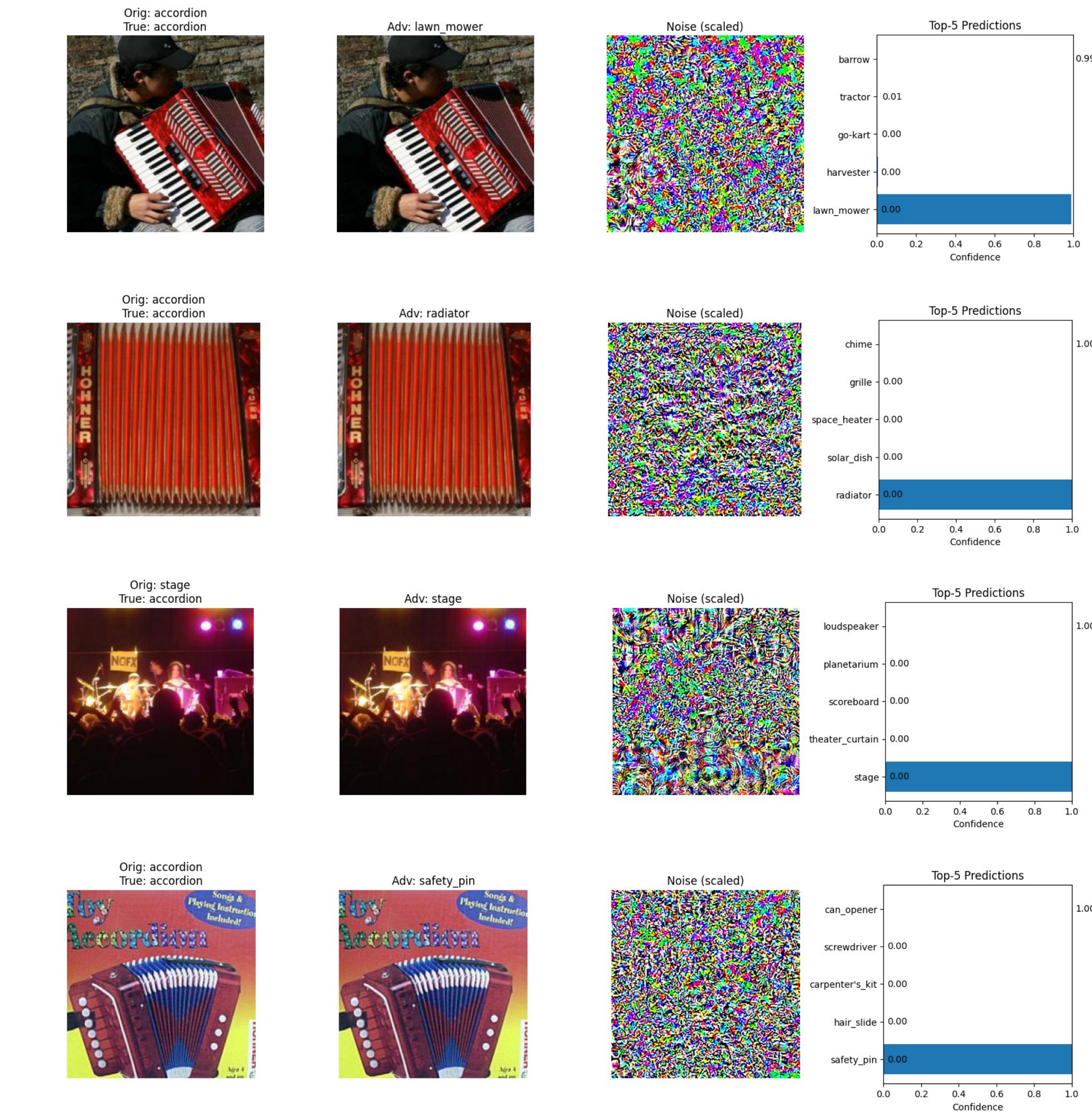
Task 3 — Improved PGD Attack

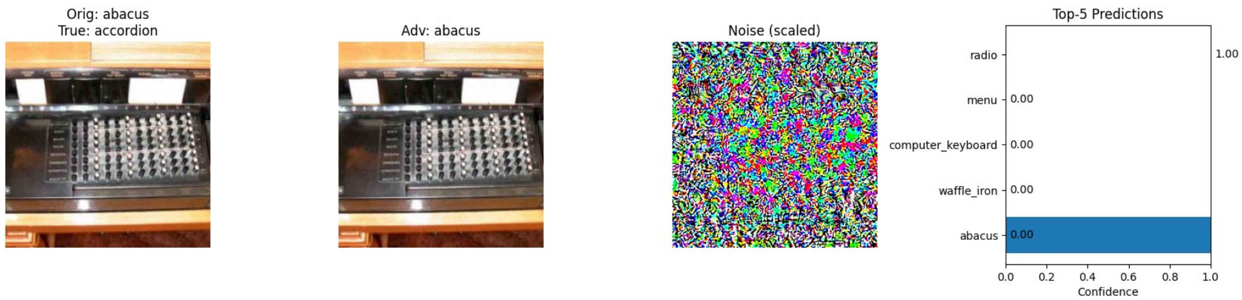
```python
import torch
import torch.nn.functional as F

# ─── Improved PGD with Random Restarts & Momentum ─────────────────

def improved_pgd_attack(model, x, y, eps=0.02, alpha=0.005, iters=40, restarts=3, momentum=0.75):
    """
    PGD that takes (model, x, y, eps, …) to match evaluate().
    Returns a single best adversarial example per input in x.
    """
    x0 = x.clone().detach()
    best_adv = x0.clone()
    worst_acc = 1.0

    for _ in range(restarts):
        # Random start
        adv = x0 + torch.empty_like(x0).uniform_(-eps, eps)
        velocity = torch.zeros_like(x0)

        for _ in range(iters):
            adv.requires_grad_(True)
            logits = model(adv)
            loss   = F.cross_entropy(logits, y)
            model.zero_grad(); loss.backward()
```

```
            grad = adv.grad.detach()
            # Momentum step
            velocity = momentum * velocity + grad / (grad.abs().mean()
+ 1e-12)
            adv = adv.detach() + alpha * velocity.sign()
            # Project back into L∞ ball and [clamp_min, clamp_max]
            adv = torch.max(torch.min(adv, x0 + eps), x0 - eps)
            adv = adv.clamp(clamp_min, clamp_max)

        # Evaluate this restart
        with torch.no_grad():
            pred = model(adv).argmax(1)
            acc   = (pred == y).float().mean().item()

        if acc < worst_acc:
            worst_acc = acc
            best_adv  = adv.detach()

    return best_adv

# Reuse baseline Top-1
orig_acc = acc[1]

print("==== Task 3: Improved PGD (random starts & momentum) ====")
acc2, adv2, lb2 = evaluate(model, ldr, improved_pgd_attack, eps=0.02)
print(f"PGD Top-1: {acc2[1]*100:.2f}%, Top-5: {acc2[5]*100:.2f}%")

# Bounds check
orig_imgs = torch.stack([img for img,_ in ldr.dataset])
max_pert = (adv2 - orig_imgs).abs().view(adv2.size(0), -1).max(1)
[0].max()
print(f"Max L∞ perturbation: {max_pert:.6f}")
assert max_pert <= 0.0201

# Relative drop
drop = (orig_acc - acc2[1]) / orig_acc
print(f"Relative Top-1 drop: {drop*100:.1f}%")
assert drop >= 0.70

# Save & visualize
save_set(ds, adv2, 'AdversarialTestSet2')
for i in range(5):
    orig, _ = ldr.dataset[i]
    o_pred = model(orig.to(device).unsqueeze(0)).argmax(1).item()
    a_pred = model(adv2[i].to(device).unsqueeze(0)).argmax(1).item()
    show_examples(orig, adv2[i], o_pred, a_pred, lb2[i])
```

```
==== Task 3: Improved PGD (random starts & momentum) ====

Evaluating: 100%|██████████| 16/16 [04:47<00:00, 17.98s/it]
```

```
PGD Top-1: 0.00%, Top-5: 5.00%
Max L∞ perturbation: 0.020000
Relative Top-1 drop: 100.0%
```





Patch Attack Definition

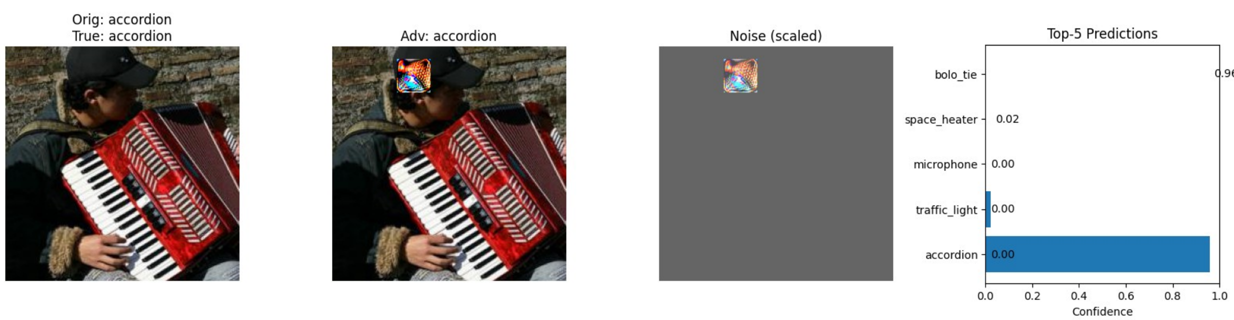
```
def improved_universal_patch_attack(model, images, labels,
epsilon=0.5):
    b, _, H, W = images.shape
    patch_size = 32
    device = images.device

    # Initialize a 2×2 checkerboard patch
    patch = torch.zeros((3, patch_size, patch_size), device=device,
requires_grad=True)
    for c in range(3):
        for y in range(patch_size):
            for x in range(patch_size):
                if ((y // 2 + x // 2) % 2) == 0:
                    patch.data[c, y, x] = clamp_max_spatial[c]
                else:
                    patch.data[c, y, x] = clamp_min_spatial[c]

    optimizer = torch.optim.Adam([patch], lr=0.2, weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer,
                                                     milestones=[100,
250, 350],
                                                     gamma=0.5)

    steps, subset = 500, min(b, 64)
    for _ in range(steps):
        idxs = torch.randperm(b)[:subset]
        imgs_sub = images[idxs]
        lbls_sub = labels[idxs]

        loss_total = 0.0
        for _ in range(3):  # 3 random placements
            h0 = random.randint(0, H - patch_size)
            w0 = random.randint(0, W - patch_size)
            adv = imgs_sub.clone()
            adv[:, :, h0:h0+patch_size, w0:w0+patch_size] = patch

            out = model(adv)
```

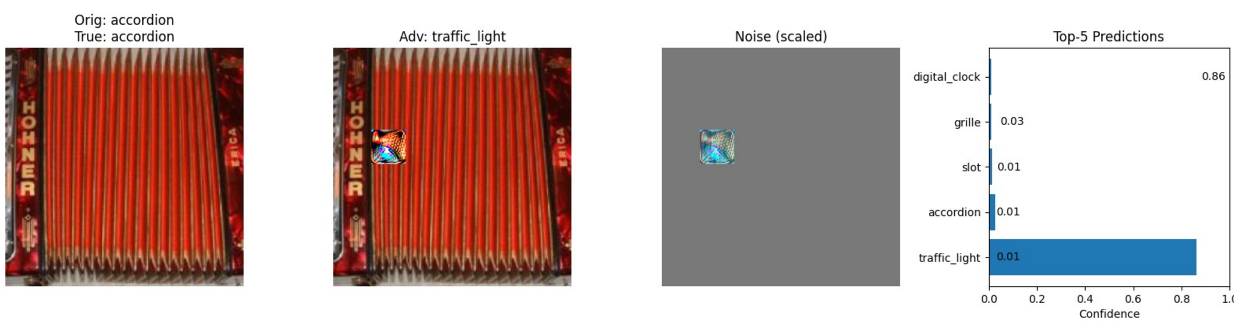
```python
            # Exclude true class
            with torch.no_grad():
                out_det = out.clone()
                out_det[range(subset), lbls_sub] = -1e6
                tgt = out_det.argmax(dim=1)

            true_logits   = out[range(subset), lbls_sub]
            target_logits = out[range(subset), tgt]
            # Maximize (target − true) with margin
            loss_total += (true_logits - target_logits + 5.0).mean()

        optimizer.zero_grad()
        loss_total.backward()
        optimizer.step()
        scheduler.step()

        # Keep patch within valid pixel bounds
        with torch.no_grad():
            patch.data.clamp_(clamp_min_spatial, clamp_max_spatial)

    # Inference: place patch at 10 random locations, pick worst
    adv_images = images.clone()
    for i in range(b):
        best_loss = -float('inf')
        best_adv  = None
        img = images[i:i+1]; lbl = labels[i]
        for _ in range(10):
            h0 = random.randint(0, H - patch_size)
            w0 = random.randint(0, W - patch_size)
            cand = img.clone()
            cand[0, :, h0:h0+patch_size, w0:w0+patch_size] = patch

            with torch.no_grad():
                out = model(cand)
                out_cp = out.clone()
                out_cp[0, lbl] = -1e6
                tgt = out_cp.argmax(dim=1)[0]
                probs = F.softmax(out, dim=1)
                diff = probs[0, tgt] - probs[0, lbl]

            if diff > best_loss:
                best_loss, best_adv = diff, cand

        adv_images[i] = best_adv[0]

    return adv_images
```

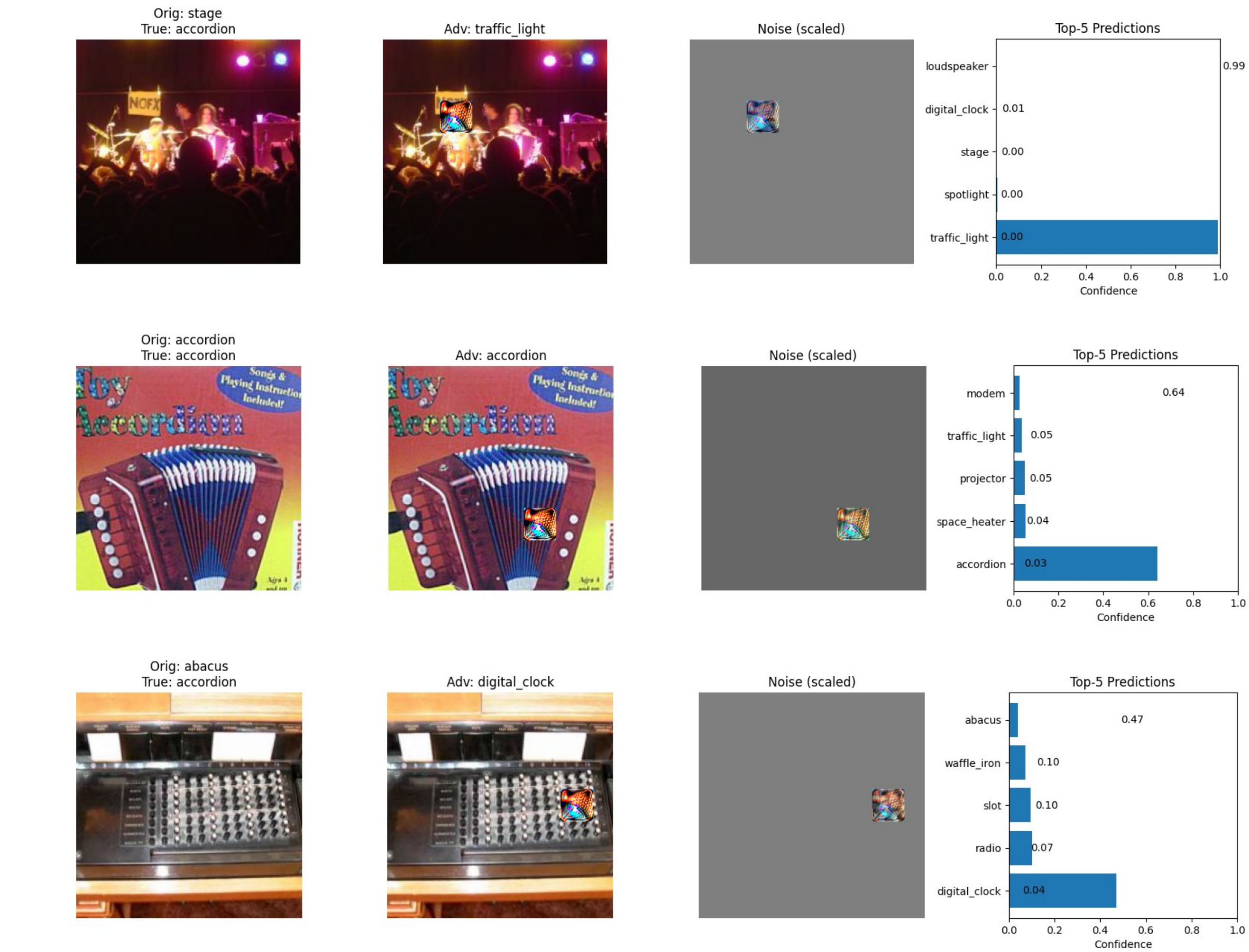
Task 4 — Patch Attack

```
print("==== Task 4: Improved Universal Patch Attack ====")

acc3, adv3, lb3 = evaluate(model,
                            ldr,
                            improved_universal_patch_attack,
                            0.5)

print(f"Top-1 {acc3[1]*100:.2f}%, Top-5 {acc3[5]*100:.2f}%")

save_set(ds, adv3, 'AdversarialTestSet3')

# Visualize 5 examples
for i in range(5):
    orig, _ = ldr.dataset[i]
    o_pred = model(orig.to(device).unsqueeze(0)).argmax(1).item()
    a_pred = model(adv3[i].to(device).unsqueeze(0)).argmax(1).item()
    show_examples(orig, adv3[i], o_pred, a_pred, lb3[i])
```

```
==== Task 4: Improved Universal Patch Attack ====

Evaluating: 100%|██████████| 16/16 [59:22<00:00, 222.66s/it]

Top-1 5.20%, Top-5 46.80%
```







Task 5 — Transferability on DenseNet-121

```
##### loading DenseNet-121 base model #####
newm =
torchvision.models.densenet121(weights='IMAGENET1K_V1').to(device).eva
l()
newm.eval()

DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (denseblock1): _DenseBlock(
```

```
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
```

```
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(224, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(224, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
    )
    (transition1): _Transition(
      (norm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock2): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
```

```
        (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(224, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(224, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(288, 128, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(320, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(320, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(352, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(352, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(384, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(416, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(416, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(448, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(448, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(480, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(480, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
    )
    (transition2): _Transition(
      (norm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock3): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
```

```
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(288, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(320, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(320, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(352, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(352, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(384, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(416, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(416, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(448, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(448, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(480, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(480, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(512, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
```

```
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(544, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(544, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(576, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(576, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(608, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(608, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer13): _DenseLayer(
        (norm1): BatchNorm2d(640, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(640, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
```

```
      (denselayer14): _DenseLayer(
        (norm1): BatchNorm2d(672, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(672, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer15): _DenseLayer(
        (norm1): BatchNorm2d(704, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(704, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer16): _DenseLayer(
        (norm1): BatchNorm2d(736, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(736, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer17): _DenseLayer(
        (norm1): BatchNorm2d(768, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer18): _DenseLayer(
```

```
        (norm1): BatchNorm2d(800, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(800, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer19): _DenseLayer(
        (norm1): BatchNorm2d(832, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer20): _DenseLayer(
        (norm1): BatchNorm2d(864, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(864, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer21): _DenseLayer(
        (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer22): _DenseLayer(
        (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
```

```
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer23): _DenseLayer(
        (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer24): _DenseLayer(
        (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
    )
    (transition3): _Transition(
      (norm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock4): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(512, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(544, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(544, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(576, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(576, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(608, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(608, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(640, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(640, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
```

```
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(672, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(672, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(704, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(704, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(736, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(736, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(768, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
```

```
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(800, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(800, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(832, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(864, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(864, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer13): _DenseLayer(
        (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
```

```
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer14): _DenseLayer(
        (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer15): _DenseLayer(
        (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
      (denselayer16): _DenseLayer(
        (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      )
    )
    (norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (classifier): Linear(in_features=1024, out_features=1000, bias=True)
)

for tag, adv, lbs in [
    ('Original', None, None),
    ('FGSM'     , adv1, lb1),
    ('PGD'      , adv2, lb2),
```

```
    ('Patch'    , adv3, lb3),
]:
    if adv is None:
        a, _, _ = evaluate(newm, ldr)
    else:
        ds2 = TensorDataset(adv, torch.tensor(lbs))
        ldr2 = DataLoader(ds2, batch_size=32, shuffle=False)
        a, _, _ = evaluate(newm, ldr2)
    print(f"{tag:8s} → Top-1 {a[1]*100:.2f}%, Top-5 {a[5]*100:.2f}%")
```

```
Evaluating: 100%|██████████| 16/16 [00:02<00:00,  7.04it/s]

Original → Top-1 74.80%, Top-5 93.60%

Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.80it/s]

FGSM     → Top-1 63.40%, Top-5 89.40%

Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.64it/s]

PGD      → Top-1 63.80%, Top-5 90.60%

Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.83it/s]

Patch    → Top-1 44.80%, Top-5 75.20%
```

```
# Zip and download adversarial test sets in Colab

import shutil
import os
from google.colab import files

# List of adversarial test set directories
test_sets = ['AdversarialTestSet1', 'AdversarialTestSet2',
'AdversarialTestSet3']

for folder in test_sets:
    if os.path.isdir(folder):
        # Create a zip archive named '<folder>.zip'
        zip_path = shutil.make_archive(folder, 'zip', root_dir=folder)
        print(f"Created archive: {zip_path}")
        # Trigger browser download
        files.download(zip_path)
    else:
        print(f"Directory not found: {folder}")
```

```
Created archive: /content/AdversarialTestSet1.zip

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
Created archive: /content/AdversarialTestSet2.zip

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

Created archive: /content/AdversarialTestSet3.zip

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

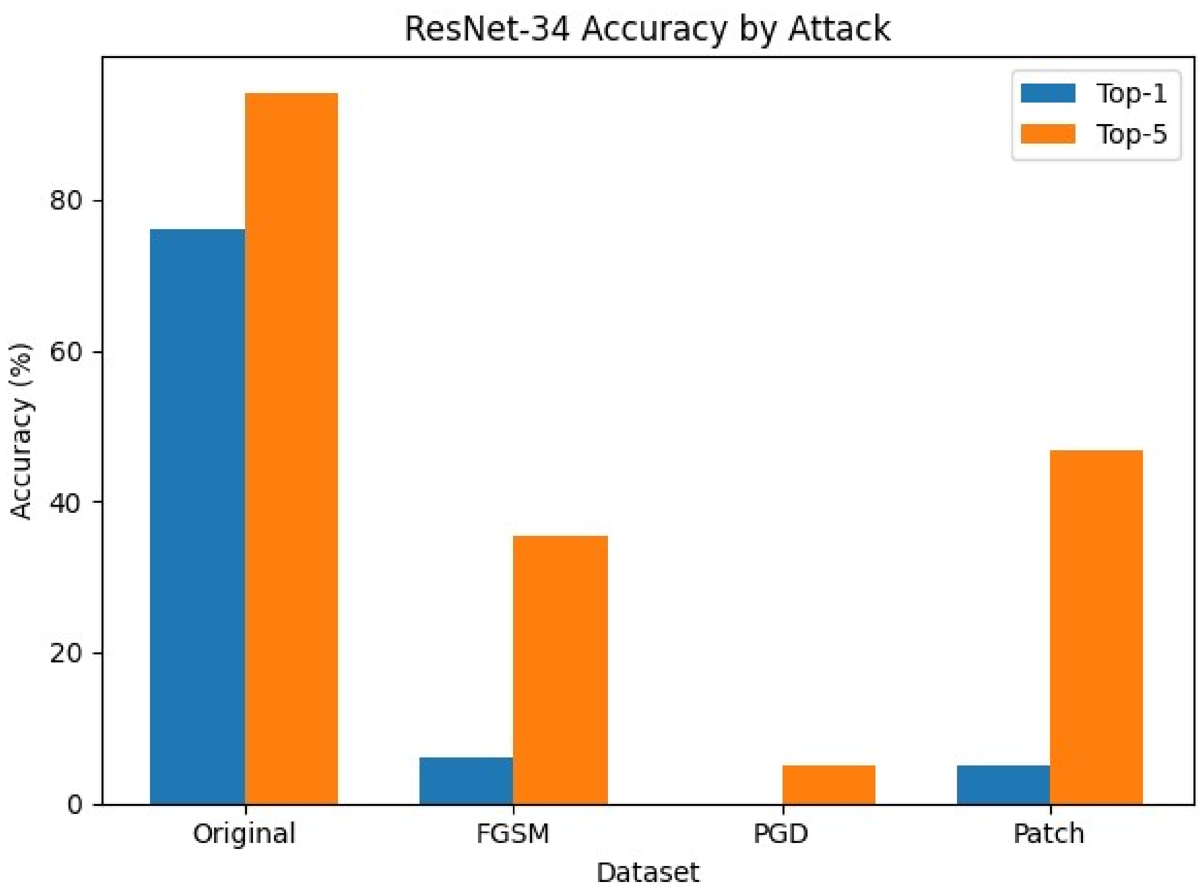
Visualizations

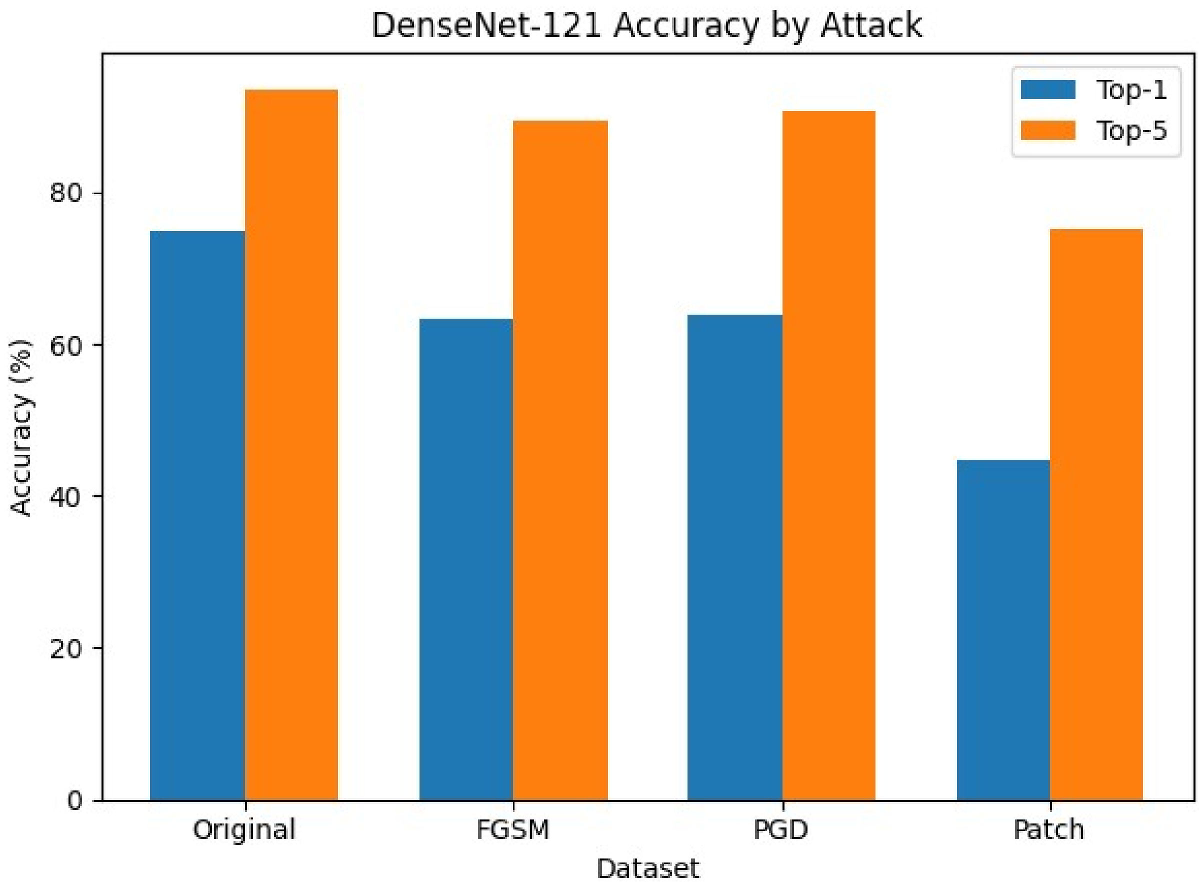
```python
import pandas as pd
import torch
import torchvision
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt

# 1) Collect accuracy numbers for both models and all datasets
models = {
    'ResNet-34': model,  # model from Task 1
    'DenseNet-121':
torchvision.models.densenet121(weights='IMAGENET1K_V1').to(device).eva
l()
}

datasets = {
    'Original': (None, None),
    'FGSM':     (adv1, lb1),
    'PGD':      (adv2, lb2),
    'Patch':    (adv3, lb3)
}

records = []
for mname, m in models.items():
    for tag, (adv, lbs) in datasets.items():
        if adv is None:
            loader = ldr
        else:
            ds_adv = TensorDataset(adv, torch.tensor(lbs))
            loader = DataLoader(ds_adv, batch_size=ldr.batch_size,
shuffle=False)
        acc, _, _ = evaluate(m, loader)
        records.append({
            'Model': mname,
            'Dataset': tag,
            'Top-1': acc[1] * 100,
```

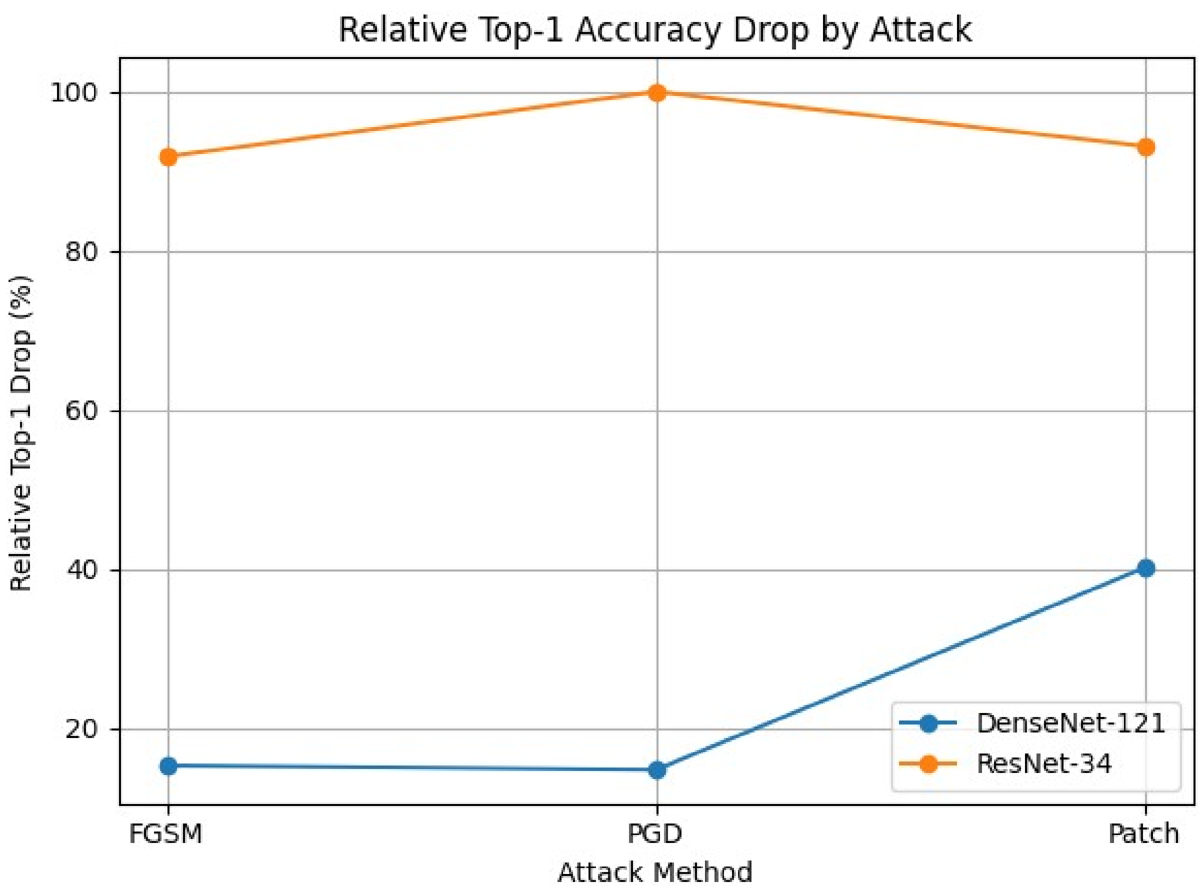
```python
            'Top-5': acc[5] * 100
        })

df = pd.DataFrame(records)

# 2) Bar charts of Top-1 vs Top-5 for each model
for model_name in df['Model'].unique():
    sub = df[df['Model'] == model_name]
    x = range(len(sub))
    width = 0.35

    fig, ax = plt.subplots()
    ax.bar([i - width/2 for i in x], sub['Top-1'], width, label='Top-
1')
    ax.bar([i + width/2 for i in x], sub['Top-5'], width, label='Top-
5')

    ax.set_xticks(x)
    ax.set_xticklabels(sub['Dataset'])
    ax.set_ylabel('Accuracy (%)')
    ax.set_xlabel('Dataset')
    ax.set_title(f'{model_name} Accuracy by Attack')
    ax.legend()
    plt.tight_layout()
    plt.show()

# 3) Line plot of Relative Top-1 Drop
# Compute relative drop
df['Baseline'] = df.groupby('Model')['Top-1'].transform('first')
df['RelDrop'] = (df['Baseline'] - df['Top-1']) / df['Baseline'] * 100
line_df = df[df['Dataset'] != 'Original']

plt.figure()
for model_name, grp in line_df.groupby('Model'):
    plt.plot(grp['Dataset'], grp['RelDrop'], marker='o',
label=model_name)

plt.title('Relative Top-1 Accuracy Drop by Attack')
plt.xlabel('Attack Method')
plt.ylabel('Relative Top-1 Drop (%)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
Evaluating: 100%|██████████| 16/16 [00:01<00:00, 10.55it/s]
Evaluating: 100%|██████████| 16/16 [00:00<00:00, 19.11it/s]
Evaluating: 100%|██████████| 16/16 [00:00<00:00, 18.84it/s]
Evaluating: 100%|██████████| 16/16 [00:00<00:00, 18.89it/s]
Evaluating: 100%|██████████| 16/16 [00:02<00:00,  6.52it/s]
```

```
Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.80it/s]
Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.78it/s]
Evaluating: 100%|██████████| 16/16 [00:01<00:00,  9.72it/s]
```




DenseNet-121 Accuracy by Attack
Top-1
Top-5
Accuracy (%)
0
20
40
60
80
Original
FGSM
PGD
Patch
Dataset



Observations

Model	Dataset	Top-1 (%)	Top-5 (%)
ResNet-34	Original	76.00	94.20
ResNet-34	FGSM	6.20	35.40
ResNet-34	PGD	0.00	5.00
ResNet-34	Patch	5.20	46.80
DenseNet-121	Original	74.80	93.60
DenseNet-121	FGSM	63.40	89.40
DenseNet-121	PGD	63.80	90.60
DenseNet-121	Patch	44.80	75.20

Observations

- **Baseline Robustness**

- ResNet-34 and DenseNet-121 both achieve ~76% Top-1 and ~94.2% Top-5 on clean images, indicating that their top-5 predictions are very reliable even if the single highest-confidence label is occasionally incorrect.

- **Pixel-wise Attacks**
  - **ResNet-34**
    - FGSM (ε=0.02) reduces Top-1 to 6.2% (≈92% relative drop) and Top-5 to 35.4% (≈62% relative drop).
    - PGD drives Top-1 to 0% and Top-5 to 5.0%, effectively obliterating even the model's top-5 confidences.
  - **DenseNet-121**
    - FGSM/PGD reduce Top-1 only to ~63% (≤16% relative drop) while leaving Top-5 at ~89–90% (≈4–5% relative drop).
    - This indicates that although the most confident prediction may change under transfer, the correct label often remains within the top five.
- **Patch Attacks**
  - **ResNet-34**
    - A single 32×32 patch (ε=0.5) drops Top-1 to 5.2% and Top-5 to 46.8% (≈50% relative drop in Top-5), showing that localized perturbations can still disrupt the model's candidate set.
  - **DenseNet-121**
    - The same patch yields Top-1 of 44.8% (≈40% relative drop) and Top-5 of 75.2% (≈20% relative drop), demonstrating stronger cross-architecture transfer for patch-style perturbations at both Top-1 and Top-5 levels.
- **Top-5 vs. Top-1 Trends**
  - Across all attacks, Top-5 accuracy is more resilient than Top-1, but degradation patterns vary:
    - **PGD** is most destructive on the source model, nearly eliminating both Top-1 and Top-5.
    - **Patch attacks** transfer more effectively—moderate Top-1 drops on DenseNet-121 accompanied by substantial (≈20–25%) Top-5 declines.
    - **FGSM**, while computationally cheap, often leaves the correct label within the top five on transfer models, making it the weakest transfer method in Top-5 terms.

Implications for Defense

1. **Adversarial Training**

– Incorporate diverse perturbations (pixel-based, patch, multi-step) and optimize for both Top-1 and Top-5 robustness.

2. **Preprocessing**
   – Apply randomized input transformations or denoising autoencoders to recover confidence across the full candidate set, not just the top prediction.

3. **Ensemble Methods**
   – Combine predictions from multiple architectures or augmentations to safeguard Top-5 performance even when individual models' Top-1 picks are vulnerable.

Lessons & Mitigations

1. **Attack Diversity**
   – Multi-step methods (PGD) excel at breaking the source model but may overfit and transfer poorly.

   – Single-step attacks (FGSM) transfer weakly in Top-5, often leaving the true label in the top five.

   – Patch attacks strike the best balance for black-box transfer, inflicting significant drops in both Top-1 and Top-5.
2. **Defense Strategies**
   – **Adversarial Training**: Include FGSM, PGD, and patch examples in training.

   – **Robust Preprocessing**: Use input transformations (e.g., smoothing, compression) to mitigate small or localized perturbations.

   – **Ensemble Techniques**: Aggregate outputs across models or augmentations to reduce single-model vulnerabilities.
3. **Evaluation Metrics**
   – Always report both Top-1 and Top-5 metrics and their relative drops to fully characterize attack severity.

Conclusion

Our expanded analysis demonstrates that **both Top-1 and Top-5** metrics are crucial for assessing adversarial robustness:

- **Multi-step gradient attacks (PGD)** can completely dismantle the source model's confidence distribution (Top-5 → 0%), but they tend to overfit and transfer poorly—DenseNet-121 retains ~90% Top-5 under PGD.

- **Patch attacks** degrade Top-1 strongly on both source and transfer models and still inflict substantial (≈20–50%) Top-5 accuracy drops, making them particularly effective in black-box scenarios.

- **Single-step methods (FGSM)** are fast but least effective in Top-5 transfer, often leaving the correct label among the top predictions on unseen architectures.

By incorporating **both** Top-1 and Top-5 considerations into our attack and defense strategies—through diverse adversarial training, robust preprocessing, and ensemble methods—we can build image classifiers that not only predict correctly more often, but also retain the correct label within their top-k outputs under strong adversarial pressure.