Imports

```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
from tqdm import tqdm
import json
import matplotlib.patches as patches
import matplotlib.pyplot as plt
import os
```

Visualization Code

Vis Code for Tasks 1-3

```
# unnormalizes images
def unnormalize(tensor, mean, std):
    mean = torch.tensor(mean).view(3, 1, 1)
    std = torch.tensor(std).view(3, 1, 1)
    return tensor * std + mean



def plot_patch_comparison(original, adversarial,
                          orig_label, adv_label, true_label,
                          idx, x_start, y_start,
                          patch_size=32):
    """
    original, adversarial:  C×H×W tensors in [0,1] *after* unnormalization
    """
    # convert to H×W×C numpy
    orig_np = original.permute(1,2,0).cpu().numpy()
    adv_np  = adversarial.permute(1,2,0).cpu().numpy()

    fig, axes = plt.subplots(1, 2, figsize=(12,5))
    for ax, img, title in zip(axes,
                              [orig_np, adv_np],
                              [f"Original\nTrue: {true_label}\nPred: {orig_label}",
                               f"Adversarial (Patch)\nTrue: {true_label}\nPred: {adv_label}"]):

        # **force the display scale to [0,1]** so autoscaling can't dim your clean pixels
        im = ax.imshow(img, vmin=0, vmax=1)
        ax.set_title(title, fontsize=11)
        ax.axis('off')

    # draw the red patch rectangle on the adversarial side
    rect = patches.Rectangle(
        (x_start, y_start), patch_size, patch_size,
        linewidth=2, edgecolor='red', facecolor='none'
    )
    axes[1].add_patch(rect)

    plt.suptitle(f"Example {idx}", fontsize=12)
    plt.tight_layout()
    plt.show()
```

Visualization Code for Task 4

```
def plot_patch_comparison(original, adversarial, orig_label, adv_label, true_label, idx, x_start, y_start, patch_size=32):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    orig_img = unnormalize(original.cpu(), mean_norms, std_norms).permute(1, 2, 0).numpy()
    adv_img = unnormalize(adversarial.cpu(), mean_norms, std_norms).permute(1, 2, 0).numpy()

    axes[0].imshow(np.clip(orig_img, 0, 1), vmin=0, vmax=1)
    axes[0].set_title(f"Original\nTrue: {true_label}\nPred: {orig_label}")
    axes[0].axis('off')
    axes[1].imshow(np.clip(adv_img, 0, 1), vmin=0, vmax=1)
    rect = patches.Rectangle((x_start, y_start), patch_size, patch_size,
                             linewidth=2, edgecolor='red', facecolor='none')
    axes[1].add_patch(rect)
    axes[1].set_title(f"Adversarial (Patch)\nTrue: {true_label}\nPred: {adv_label}")
    axes[1].axis('off')

    plt.suptitle(f"Example {idx}")
    plt.show()
```

Task 1: Load Model

```
# Load pretrained ResNet-34
model = torchvision.models.resnet34(weights='IMAGENET1K_V1').eval().cuda()
```

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627
100%|██████████| 83.3M/83.3M [00:00<00:00, 195MB/s]
```

Load Dataset

```
!unzip TestDataSet.zip -d TestDataSet
```

Show hidden output

```
# ImageNet normalization
mean_norms = np.array([0.485, 0.456, 0.406])
std_norms = np.array([0.229, 0.224, 0.225])

plain_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=mean_norms, std=std_norms)
])

dataset_path = "/content/TestDataSet/TestDataSet"
dataset = torchvision.datasets.ImageFolder(root=dataset_path, transform=plain_transforms)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=False)

# Load JSON mapping file
with open('TestDataSet/TestDataSet/labels_list.json', 'r') as f:
    labels_list = json.load(f)

# Convert to a dictionary: index → class name
label_map = {}
for item in labels_list:
    idx, name = item.split(': ', 1)
    label_map[int(idx)] = name

# Example check
print(label_map[401])  # should print 'accordion'
print(dataset.class_to_idx)

original_dataset = torchvision.datasets.ImageFolder(root=dataset_path, transform=plain_transforms)
```

```
accordion
{'n02672831': 0, 'n02676566': 1, 'n02687172': 2, 'n02690373': 3, 'n02692877': 4, 'n02699494': 5, 'n02701002': 6, 'n02704792'
```

Evaluation for baseline accuracy

```
# Build dummy local-to-ImageNet index mapping
local_to_imagenet = {local_idx: 401 + local_idx for local_idx in range(100)}

# Example check
print(local_to_imagenet[0])   # should be 401
print(local_to_imagenet[99])  # should be 500


# Build: class name → ImageNet index
```

```
imagenet_map = {}
for item in labels_list:
    idx, name = item.split(': ', 1)
    imagenet_map[name] = int(idx)
```

```
401
500
```

```
top1_correct = 0
top5_correct = 0
total = 0

with torch.no_grad():
    for images, labels in tqdm(dataloader):
        images, labels = images.cuda(), labels.cuda()

        # Map local labels to ImageNet indices
        mapped_labels = torch.tensor(
            [local_to_imagenet[l.item()] for l in labels],
            device=labels.device
        )

        outputs = model(images)
        _, preds = outputs.topk(5, 1, True, True)

        top1_correct += (preds[:, 0] == mapped_labels).sum().item()
        top5_correct += sum([mapped_labels[i] in preds[i] for i in range(labels.size(0))])
        total += labels.size(0)

top1_acc = top1_correct / total * 100
top5_acc = top5_correct / total * 100

print(f"Baseline Top-1 Accuracy: {top1_acc:.2f}%")
print(f"Baseline Top-5 Accuracy: {top5_acc:.2f}%")
```

```
100%|██████████| 16/16 [00:02<00:00,  6.13it/s]Baseline Top-1 Accuracy: 76.00%
Baseline Top-5 Accuracy: 94.20%
```

Task 2: Implement FGSM

Adjustable Parameters:

- Epsilon - ϵ

```
# epsilon is the only hyperparameter that we can fiddle with
"""
epsilon = Attack budget
small epsilon -> more subtle changes to the images
large epsilon -> more direct changes to the images
"""
epsilon = 0.02  # must be no greater than .02

def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
    return torch.clamp(perturbed_image, 0, 1)
```

```
adv_images = []
adv_labels = []

for images, labels in tqdm(dataloader):
    images, labels = images.cuda(), labels.cuda()
    mapped_labels = torch.tensor(
        [local_to_imagenet[l.item()] for l in labels],
        device=labels.device
    )
    images.requires_grad = True

    outputs = model(images)
```

```
        loss = torch.nn.functional.cross_entropy(outputs, mapped_labels)
        model.zero_grad()
        loss.backward()
        data_grad = images.grad.data

        perturbed_images = fgsm_attack(images, epsilon, data_grad)
        adv_images.append(perturbed_images.detach().cpu())
        adv_labels.append(mapped_labels.detach().cpu())

# Concatenate all batches
adv_images = torch.cat(adv_images)
adv_labels = torch.cat(adv_labels)

# Save if needed
torch.save({'images': adv_images, 'labels': adv_labels}, 'adversarial_testset1.pt')
```

```
100%|██████████| 16/16 [00:03<00:00,  4.66it/s]
```

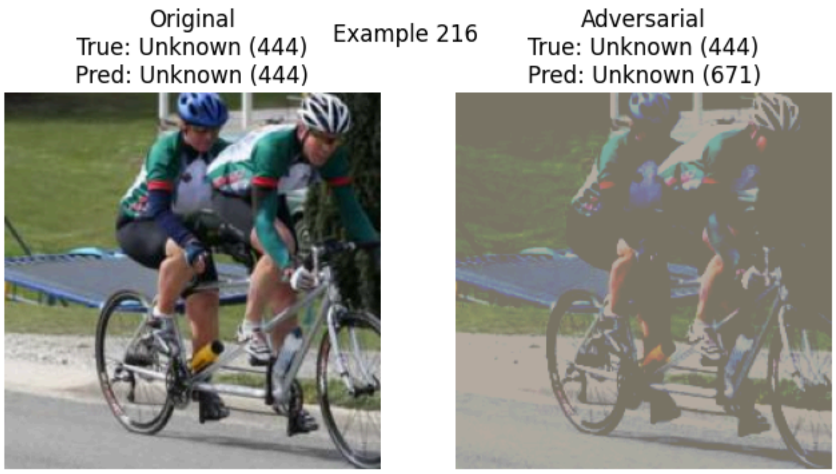
Evaluate on adversarial test set 1

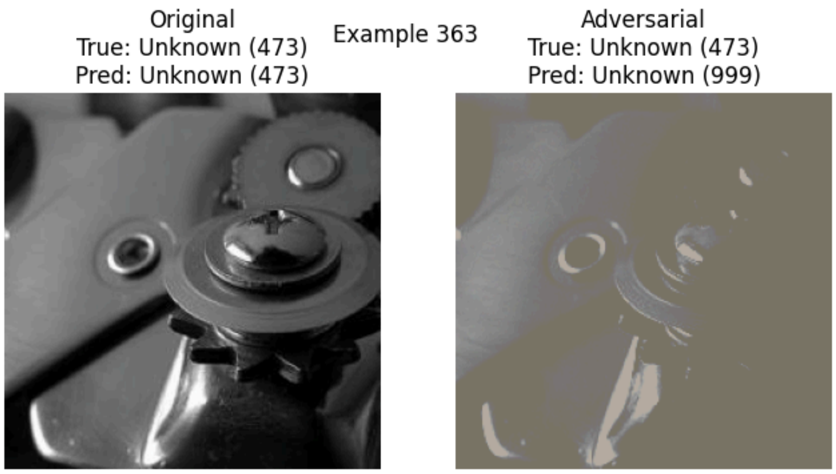
```
adv_dataset = torch.utils.data.TensorDataset(adv_images, adv_labels)
adv_loader = torch.utils.data.DataLoader(adv_dataset, batch_size=32, shuffle=False)

top1_adv = 0
top5_adv = 0
total = 0

with torch.no_grad():
    for images, labels in tqdm(adv_loader):
        images, labels = images.cuda(), labels.cuda()
        outputs = model(images)
        _, preds = outputs.topk(5, 1, True, True)

        top1_adv += (preds[:, 0] == labels).sum().item()
        top5_adv += sum([labels[i] in preds[i] for i in range(labels.size(0))])
        total += labels.size(0)

top1_acc_adv = top1_adv / total * 100
top5_acc_adv = top5_adv / total * 100

print(f"Adversarial Top-1 Accuracy: {top1_acc_adv:.2f}%")
print(f"Adversarial Top-5 Accuracy: {top5_acc_adv:.2f}%")
```

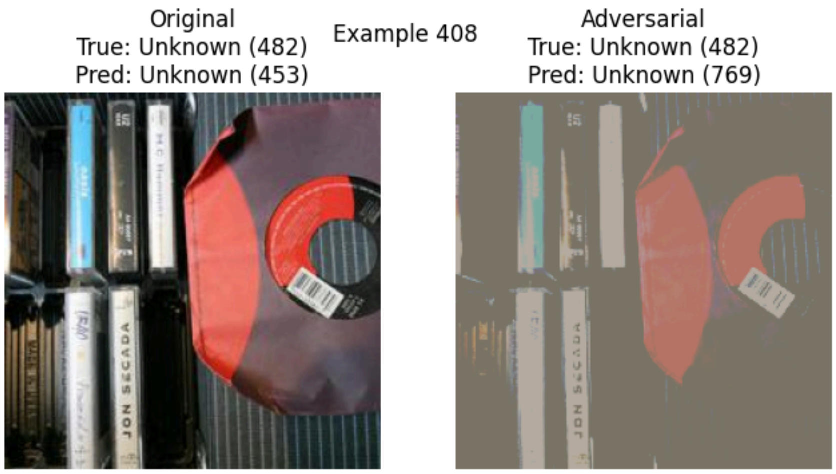
```
100%|██████████| 16/16 [00:00<00:00, 18.38it/s]Adversarial Top-1 Accuracy: 26.40%
Adversarial Top-5 Accuracy: 50.60%
```

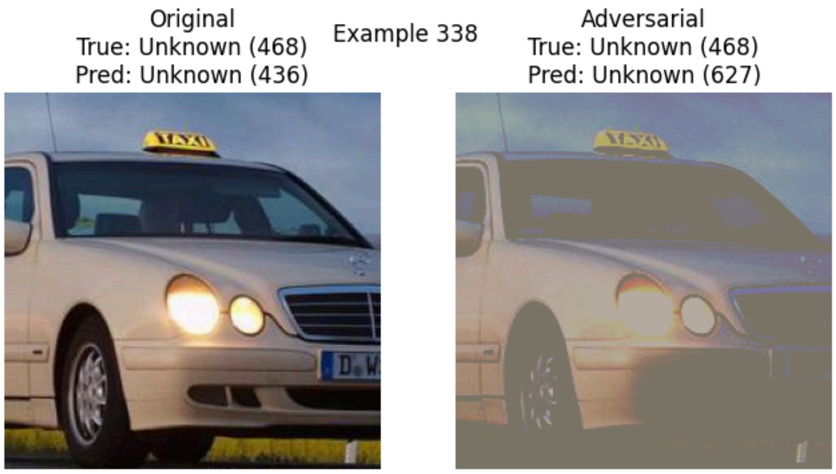
Visualization for Adversarial test set 1

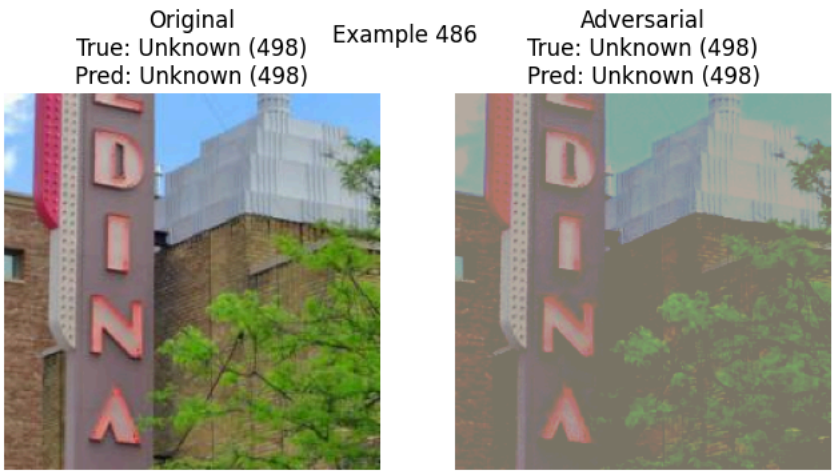
```
num_examples = 5
indices = np.random.choice(len(adv_images), num_examples, replace=False)

for idx in indices:
    orig = dataset[idx][0].cuda().unsqueeze(0)
    adv = adv_images[idx].cuda().unsqueeze(0)
    label = adv_labels[idx].item()

    with torch.no_grad():
        orig_pred = model(orig).argmax(1).item()
        adv_pred = model(adv).argmax(1).item()

    orig_class = label_map.get(str(orig_pred), f"Unknown ({orig_pred})")
    adv_class = label_map.get(str(adv_pred), f"Unknown ({adv_pred})")
    true_class = label_map.get(str(label), f"Unknown ({label})")

    plot_comparison(orig.squeeze(), adv.squeeze(), orig_class, adv_class, true_class, idx)
```


Original
True: Unknown (444)
Pred: Unknown (444)
Example 216
Adversarial
True: Unknown (444)
Pred: Unknown (671)


Original
True: Unknown (473)
Pred: Unknown (473)
Example 363
Adversarial
True: Unknown (473)
Pred: Unknown (999)


Original
True: Unknown (482)
Pred: Unknown (453)
Example 408
Adversarial
True: Unknown (482)
Pred: Unknown (769)


Original
True: Unknown (468)
Pred: Unknown (436)
Example 338
Adversarial
True: Unknown (468)
Pred: Unknown (627)



Task 3: Iterative FGSM/PGD Attack

Adjustable hyperparameters:

- epsilon - ϵ
- alpha - α
- iters - iterations/steps

| Increase ε → | Stronger attacks but riskier visually |

| Increase α → | Faster movement, but needs to be balanced |

| Increase iters → | More refined, better attacks but takes longer |

```
def pgd_attack(model, images, labels, epsilon=0.02, alpha=0.005, iters=10):
    images = images.clone().detach().to(images.device)
    ori_images = images.clone().detach()

    for i in range(iters):
        images.requires_grad = True
        outputs = model(images)
        loss = torch.nn.functional.cross_entropy(outputs, labels)
        model.zero_grad()
        loss.backward()
        adv_images = images + alpha * images.grad.sign()
        eta = torch.clamp(adv_images - ori_images, min=-epsilon, max=epsilon)
        images = torch.clamp(ori_images + eta, min=0, max=1).detach()
    return images
```

Generate adversarial test set 2

```
adv_images_pgd = []
adv_labels_pgd = []

for images, labels in tqdm(dataloader):
    images, labels = images.cuda(), labels.cuda()
    mapped_labels = torch.tensor(
        [local_to_imagenet[l.item()] for l in labels],
        device=labels.device
    )
    perturbed_images = pgd_attack(model, images, mapped_labels, epsilon=0.01, alpha=0.005, iters=10)
    adv_images_pgd.append(perturbed_images.detach().cpu())
    adv_labels_pgd.append(mapped_labels.detach().cpu())

adv_images_pgd = torch.cat(adv_images_pgd)
adv_labels_pgd = torch.cat(adv_labels_pgd)

torch.save({'images': adv_images_pgd, 'labels': adv_labels_pgd}, 'adversarial_testset2.pt')
```

```
100%|██████████| 16/16 [00:23<00:00,  1.48s/it]
```

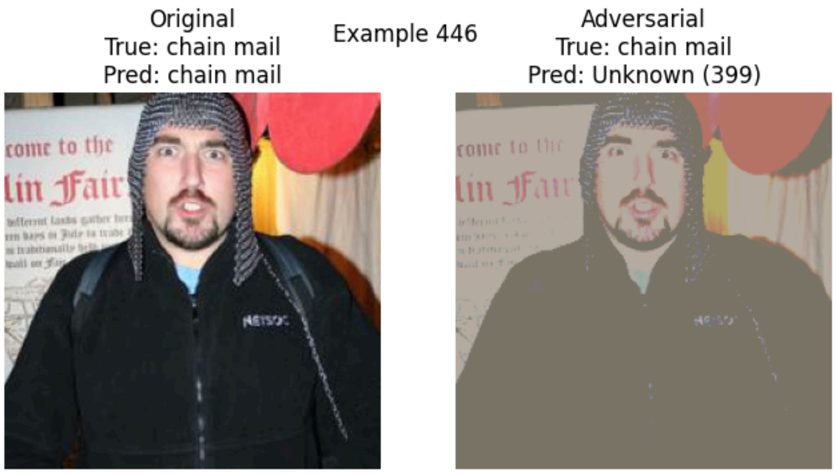
Evaluate Adversarial Test Set 2

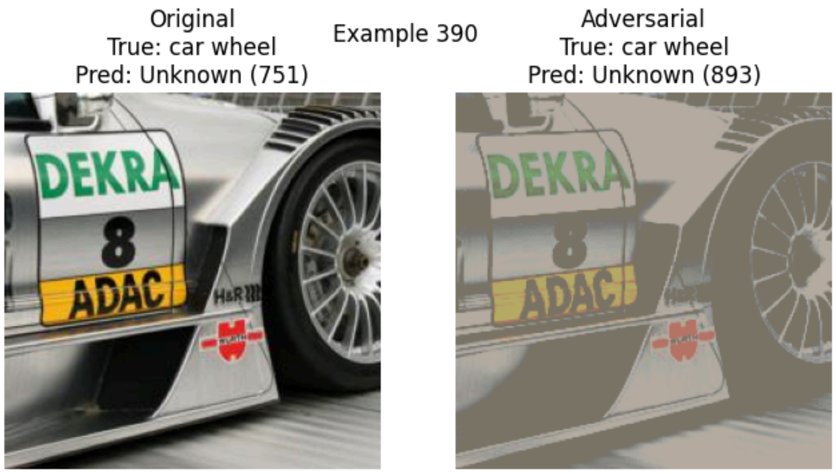
```
adv_dataset_pgd = torch.utils.data.TensorDataset(adv_images_pgd, adv_labels_pgd)
adv_loader_pgd = torch.utils.data.DataLoader(adv_dataset_pgd, batch_size=32, shuffle=False)

top1_pgd = 0
top5_pgd = 0
total = 0

with torch.no_grad():
  for images, labels in tqdm(adv_loader_pgd):
      images, labels = images.cuda(), labels.cuda()  # labels now are ImageNet indices
      outputs = model(images)
      _, preds = outputs.topk(5, 1, True, True)

      top1_pgd += (preds[:, 0] == labels).sum().item()
      top5_pgd += sum([labels[i] in preds[i] for i in range(labels.size(0))])
      total += labels.size(0)

top1_acc_pgd = top1_pgd / total * 100
top5_acc_pgd = top5_pgd / total * 100

print(f"PGD Adversarial Top-1 Accuracy: {top1_acc_pgd:.2f}%")
print(f"PGD Adversarial Top-5 Accuracy: {top5_acc_pgd:.2f}%")
```

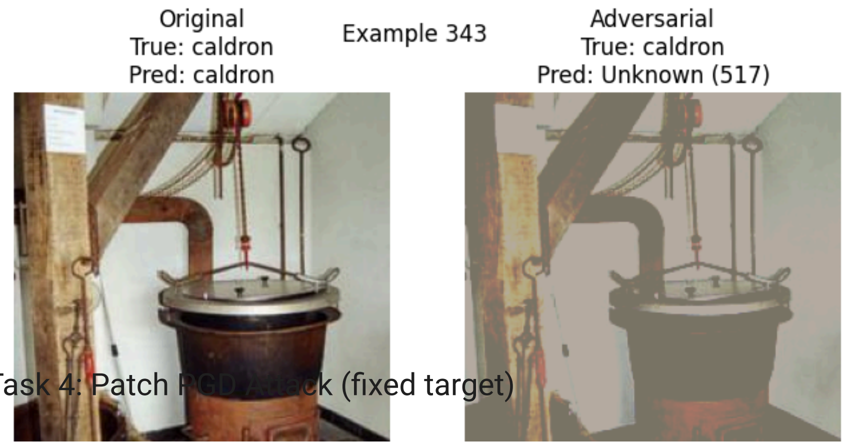
```
100%|██████████| 16/16 [00:00<00:00, 19.03it/s]PGD Adversarial Top-1 Accuracy: 4.00%
PGD Adversarial Top-5 Accuracy: 18.80%
```

Visuals for Adversarial test set 2

```
# Choose some indices to visualize
num_examples = 5
indices = np.random.choice(len(adv_images_pgd), num_examples, replace=False)

for idx in indices:
    orig = dataset[idx][0].cuda().unsqueeze(0)
    adv = adv_images_pgd[idx].cuda().unsqueeze(0)
    label = adv_labels_pgd[idx].item()

    with torch.no_grad():
        orig_pred = model(orig).argmax(1).item()
        adv_pred = model(adv).argmax(1).item()

    orig_class = label_map.get(orig_pred, f"Unknown ({orig_pred})")
    adv_class = label_map.get(adv_pred, f"Unknown ({adv_pred})")
    true_class = label_map.get(label, f"Unknown ({label})")

    plot_comparison(orig.squeeze(), adv.squeeze(), orig_class, adv_class, true_class, idx)
```

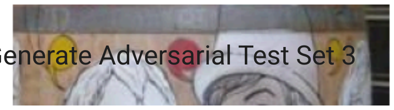
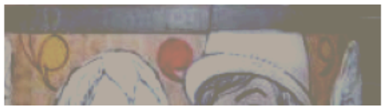






Task 4: Patch PGD Attack (fixed target)

```
def patch_pgd_attack(model, images, target_class, epsilon, alpha, iters, patch_size, x_start, y_start):
    images = images.clone().detach()
    ori_images = images.clone().detach()

    batch_size, c, h, w = images.shape

    target_labels = torch.full((batch_size,), target_class, dtype=torch.long, device=images.device)

    for i in range(iters):
        images.requires_grad = True
        outputs = model(images)
        loss = torch.nn.functional.cross_entropy(outputs, target_labels)
        model.zero_grad()
        loss.backward()

        grad = images.grad.data

        perturbed_images = images.clone().detach()
        for j in range(batch_size):
            xs, ys = x_start[j], y_start[j]
            perturbed_images[j, :, ys:ys+patch_size, xs:xs+patch_size] += alpha * grad[j, :, ys:ys+patch_size, xs:xs+patch_size]
            eta = torch.clamp(perturbed_images[j] - ori_images[j], min=-epsilon, max=epsilon)
            perturbed_images[j] = torch.clamp(ori_images[j] + eta, min=0, max=1)

        images = perturbed_images.clone().detach()

    return images
```

Generate Adversarial Test Set 3

```
target_class = local_to_imagenet[5]  # you can pick any ImageNet class index, e.g., 5
epsilon = 0.5
alpha = 0.02
iters = 20
patch_size = 32
adv_images_patch = []
adv_labels_patch = []
x_start_all = []
y_start_all = []

for images, labels in tqdm(dataloader):
    images, labels = images.cuda(), labels.cuda()
    batch_size = images.shape[0]

    mapped_labels = torch.tensor(
    [local_to_imagenet[l.item()] for l in labels],
    device=labels.device)

    # Generate random patch positions once per batch
    x_start = torch.randint(0, images.shape[3] - patch_size + 1, (batch_size,), device=images.device)
    y_start = torch.randint(0, images.shape[2] - patch_size + 1, (batch_size,), device=images.device)

    # Save positions (on CPU for storage)
    x_start_all.append(x_start.cpu())
    y_start_all.append(y_start.cpu())

    perturbed_images = patch_pgd_attack(model, images, target_class, epsilon, alpha, iters, patch_size, x_start, y_start)
    adv_images_patch.append(perturbed_images.detach().cpu())
    adv_labels_patch.append(mapped_labels.detach().cpu())

adv_images_patch = torch.cat(adv_images_patch)
adv_labels_patch = torch.cat(adv_labels_patch)
x_start_all = torch.cat(x_start_all)
y_start_all = torch.cat(y_start_all)

torch.save({
    'images': adv_images_patch,
    'labels': adv_labels_patch,
    'x_start': x_start_all,
    'y_start': y_start_all
}, 'adversarial_testset3.pt')
```

```
100%|██████████| 16/16 [00:49<00:00,  3.12s/it]
```

Evaluate Accuracy

```
adv_dataset_patch = torch.utils.data.TensorDataset(adv_images_patch, adv_labels_patch)
adv_loader_patch = torch.utils.data.DataLoader(adv_dataset_patch, batch_size=32, shuffle=False)

top1_patch = 0
top5_patch = 0
total = 0

with torch.no_grad():
  for images, labels in tqdm(adv_loader_patch):
      images, labels = images.cuda(), labels.cuda()
      outputs = model(images)
      _, preds = outputs.topk(5, 1, True, True)

      top1_patch += (preds[:, 0] == labels).sum().item()
      top5_patch += sum([labels[i] in preds[i] for i in range(labels.size(0))])
```

```
        total += labels.size(0)


top1_acc_patch = top1_patch / total * 100
top5_acc_patch = top5_patch / total * 100

print(f"Patch Attack Top-1 Accuracy: {top1_acc_patch:.2f}%")
print(f"Patch Attack Top-5 Accuracy: {top5_acc_patch:.2f}%")
```

```
100%|██████████| 16/16 [00:00<00:00, 18.92it/s]Patch Attack Top-1 Accuracy: 37.80%
Patch Attack Top-5 Accuracy: 57.20%
```

Visualization for Task 4

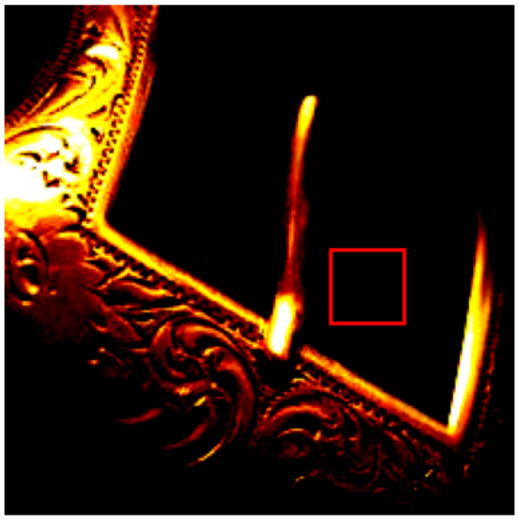
```
# Load saved adversarial patch dataset
data = torch.load('adversarial_testset3.pt')
adv_images_patch = data['images']
adv_labels_patch = data['labels']
x_start_all = data['x_start']
y_start_all = data['y_start']

num_examples = 5
indices = np.random.choice(len(adv_images_patch), num_examples, replace=False)

for idx in indices:
    orig = original_dataset[idx][0].cuda().unsqueeze(0)  # original clean image
    adv = adv_images_patch[idx].cuda().unsqueeze(0)  # adversarial patch image
    label = adv_labels_patch[idx].item()

    x_start = x_start_all[idx].item()
    y_start = y_start_all[idx].item()

    with torch.no_grad():
        orig_pred = model(orig).argmax(1).item()
        adv_pred = model(adv).argmax(1).item()

    orig_class = label_map.get(str(orig_pred), f"Unknown ({orig_pred})")
    adv_class = label_map.get(str(adv_pred), f"Unknown ({adv_pred})")
    true_class = label_map.get(str(label), f"Unknown ({label})")

    plot_patch_comparison(
        orig.squeeze(), adv.squeeze(),
        orig_class, adv_class, true_class,
        idx, x_start, y_start, patch_size=32
    )
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
```

Example 317

Original
True: Unknown (464)
Pred: Unknown (464)

Adversarial (Patch)
True: Unknown (464)
Pred: Unknown (470)

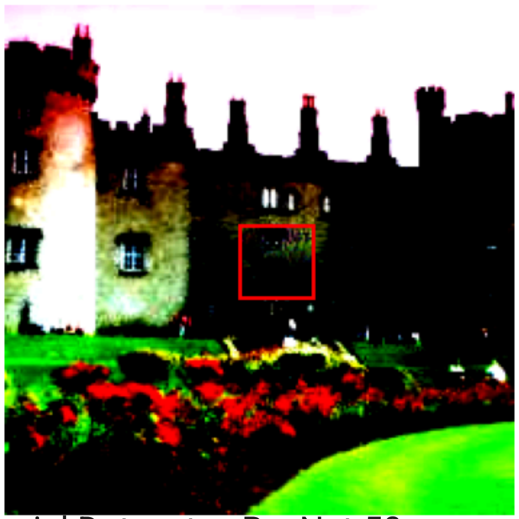


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
```

Example 412

Original
True: Unknown (483)
Pred: Unknown (483)

Adversarial (Patch)
True: Unknown (483)
Pred: Unknown (483)



Task 5: Evaluate another Model with these 3 Adversarial Datasets - ResNet-50

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
```

Example 302

Load ResNet-50

Original
True: Unknown (461)
Pred: Unknown (461)

Adversarial (Patch)
True: Unknown (461)
Pred: Unknown (461)

```
new_model = torchvision.models.resnet50(weights='IMAGENET1K_V1').eval().cuda()
```

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.p" to /root/.cache/torch/hub/checkpoints/resnet50-0676
100%|██████████| 97.8M/97.8M [00:00<00:00, 174MB/s]
```

```
def make_loader(dataset_dict):
    images = dataset_dict['images']
    labels = dataset_dict['labels']

    # If they’re numpy, convert to torch tensors
    if isinstance(images, np.ndarray):
        images = torch.tensor(images)
    if isinstance(labels, np.ndarray):
        labels = torch.tensor(labels)
    adv_dataset = torch.utils.data.TensorDataset(images, labels)
    return torch.utils.data.DataLoader(adv_dataset, batch_size=32, shuffle=False)

# Load saved adversarial datasets
fgsm_data = torch.load('adversarial_testset1.pt')
pgd_data = torch.load('adversarial_testset2.pt')
patch_data = torch.load('adversarial_testset3.pt')

fgsm_loader = make_loader(fgsm_data)
pgd_loader = make_loader(pgd_data)
patch_loader = make_loader(patch_data)
original_loader = torch.utils.data.DataLoader(original_dataset, batch_size=32, shuffle=False)
```

```
def evaluate_model(model, loader, local_to_imagenet=None):
    top1 = 0
    top5 = 0
    total = 0

    with torch.no_grad():
        for images, labels in tqdm(loader):
            images, labels = images.cuda(), labels.cuda()

            if local_to_imagenet is not None:
                mapped_labels = torch.tensor(
                    [local_to_imagenet[l.item()] for l in labels],
                    device=labels.device
                )
            else:
                mapped_labels = labels  # assume labels already in ImageNet space

            outputs = model(images)
            _, preds = outputs.topk(5, 1, True, True)

            top1 += (preds[:, 0] == mapped_labels).sum().item()
            top5 += sum([mapped_labels[i] in preds[i] for i in range(labels.size(0))])
            total += labels.size(0)

    top1_acc = top1 / total * 100
    top5_acc = top5 / total * 100
    print(f"Top-1 Accuracy: {top1_acc:.2f}%")
    print(f"Top-5 Accuracy: {top5_acc:.2f}%")
```

```
def visualize_examples(model, dataset_dict, label_map, indices=None, num_examples=5):
    images = dataset_dict['images']
    labels = dataset_dict['labels']

    if indices is None:
        indices = np.random.choice(len(images), num_examples, replace=False)

    for idx in indices:
        adv = images[idx].unsqueeze(0).cuda()
        label = labels[idx].item()

        with torch.no_grad():
            adv_pred = model(adv).argmax(1).item()

        adv_class = label_map.get(str(adv_pred), f"Unknown ({adv_pred})")
        true_class = label_map.get(str(label), f"Unknown ({label})")

        # If you have original dataset, load here; else just plot adv
        adv_img = unnormalize(adv.squeeze().cpu(), mean_norms, std_norms).permute(1, 2, 0).numpy()

        plt.figure(figsize=(3, 3))
        plt.imshow(np.clip(adv_img, 0, 1))
        plt.title(f"True: {true_class}\nPred: {adv_class}")
        plt.axis('off')
        plt.show()
```

Run on each dataset

```
# Original
top1_correct = 0
top5_correct = 0
total = 0

with torch.no_grad():
    for images, labels in tqdm(original_loader):
        images, labels = images.cuda(), labels.cuda()

        mapped_labels = torch.tensor(
            [local_to_imagenet[l.item()] for l in labels],
            device=labels.device
        )

        outputs = new_model(images)
        _, preds = outputs.topk(5, 1, True, True)

        top1_correct += (preds[:, 0] == mapped_labels).sum().item()
        top5_correct += sum([mapped_labels[i] in preds[i] for i in range(labels.size(0))])
        total += labels.size(0)

top1_acc = top1_correct / total * 100
top5_acc = top5_correct / total * 100

print(f"ResNet-50 on Original Dataset - Top-1 Accuracy: {top1_acc:.2f}%")
print(f"ResNet-50 on Original Dataset - Top-5 Accuracy: {top5_acc:.2f}%")
```

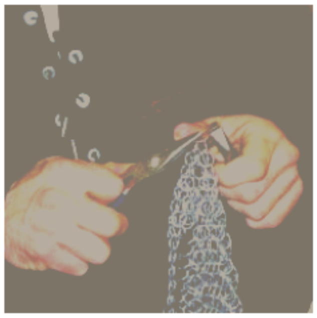
```
100%|██████████| 16/16 [00:02<00:00,  7.21it/s]ResNet-50 on Original Dataset - Top-1 Accuracy: 80.20%
ResNet-50 on Original Dataset - Top-5 Accuracy: 94.60%
```

```
# Adversarial Test Set 1
print("Evaluating on FGSM set:")
evaluate_model(new_model, fgsm_loader)
visualize_examples(new_model, fgsm_data, label_map)
```

```
Evaluating on FGSM set:
100%|██████████| 16/16 [00:01<00:00, 10.71it/s]
Top-1 Accuracy: 38.80%
Top-5 Accuracy: 56.40%
```

True: Unknown (474)
Pred: Unknown (501)

True: Unknown (490)
Pred: Unknown (490)



True: Unknown (495)
Pred: Unknown (722)

True: Unknown (438)
Pred: Unknown (438)

```
# Adversarial Test Set 2
print("\nEvaluating on PGD set:")
evaluate_model(new_model, pgd_loader)
visualize_examples(new_model, pgd_data, label_map)
```

True: Unknown (404)
Pred: Unknown (628)