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



# Plot Examples
def plot_comparison(original, adversarial, orig_label, adv_label, true_label, idx):
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))

    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))
    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))
    axes[1].set_title(f"Adversarial\nTrue: {true_label}\nPred: {adv_label}")
    axes[1].axis('off')

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

Visualization Code for Task 4

```
def plot_patch_comparison(original, adversarial, orig_label, adv_label, true_label, idx, x_s
    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))
    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))
    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()
```

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_trans
```

```
accordion
{'n02672831': 0, 'n02676566': 1, 'n02687172': 2, 'n02690373': 3, 'n02692877': 4, 'n02699
```

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,  6.55it/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,  4.45it/s]
```

Evaluate on adversarial test set 1

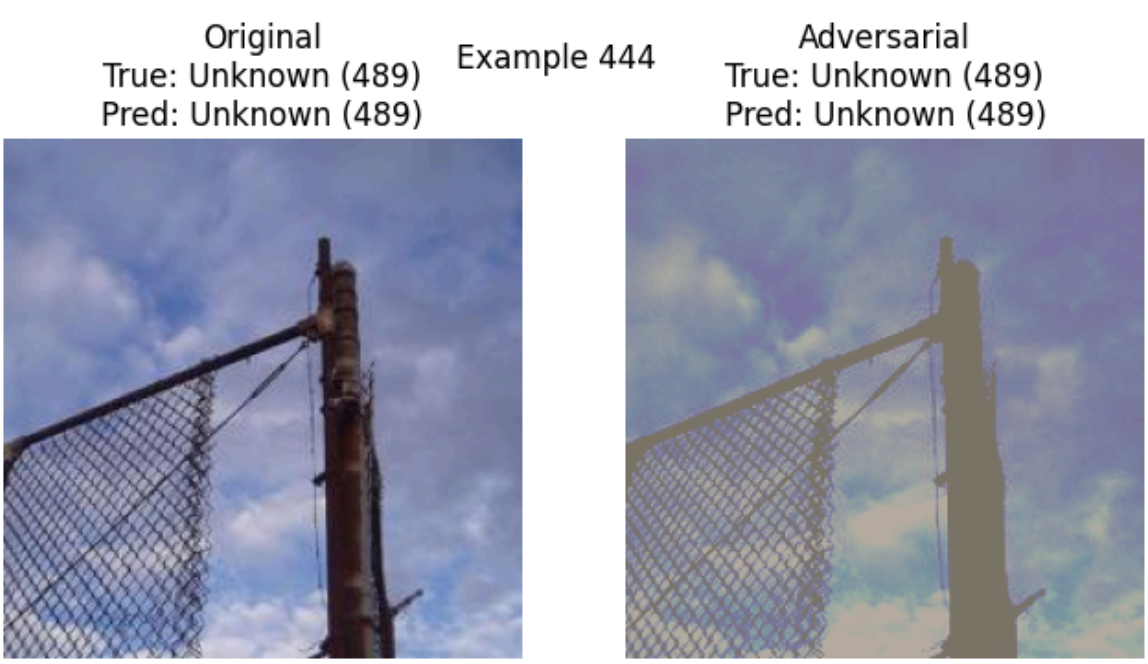
```
adv_dataset = torch.utils.data.TensorDataset(adv_images, adv_labels)
adv_loader = torch.utils.data.DataLoader(adv_dataset, batch_size=32, shuffle=False)

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}%")
```

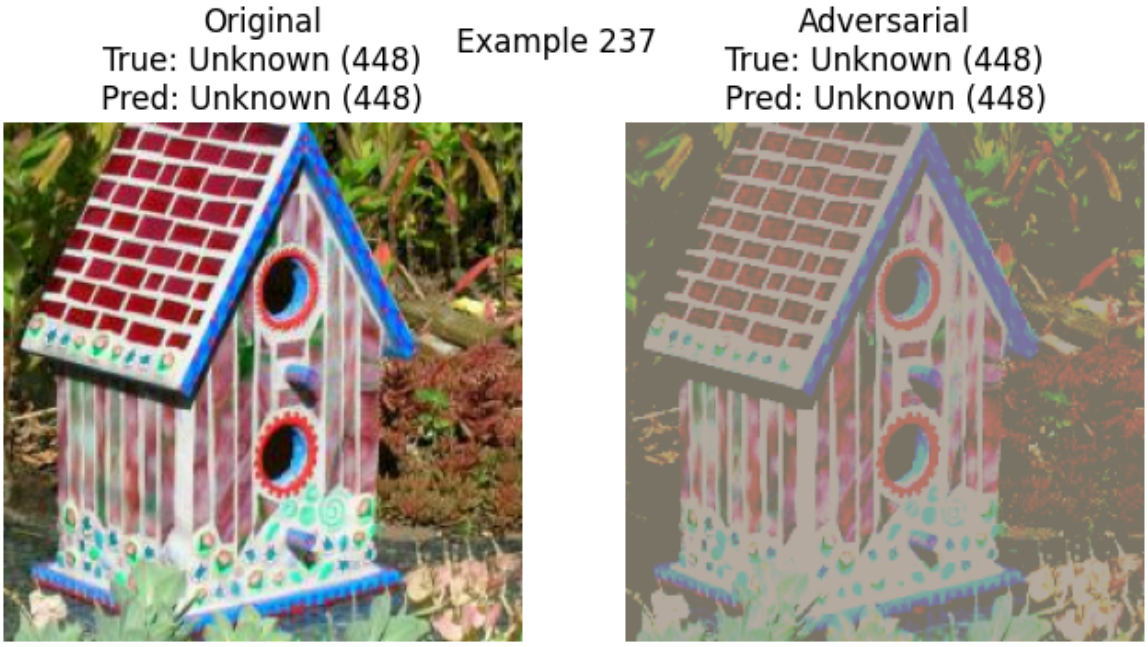
```
100%|██████████| 16/16 [00:00<00:00, 18.99it/s]Adversarial Top-1 Accuracy: 26.40%
Adversarial Top-5 Accuracy: 50.60%
```

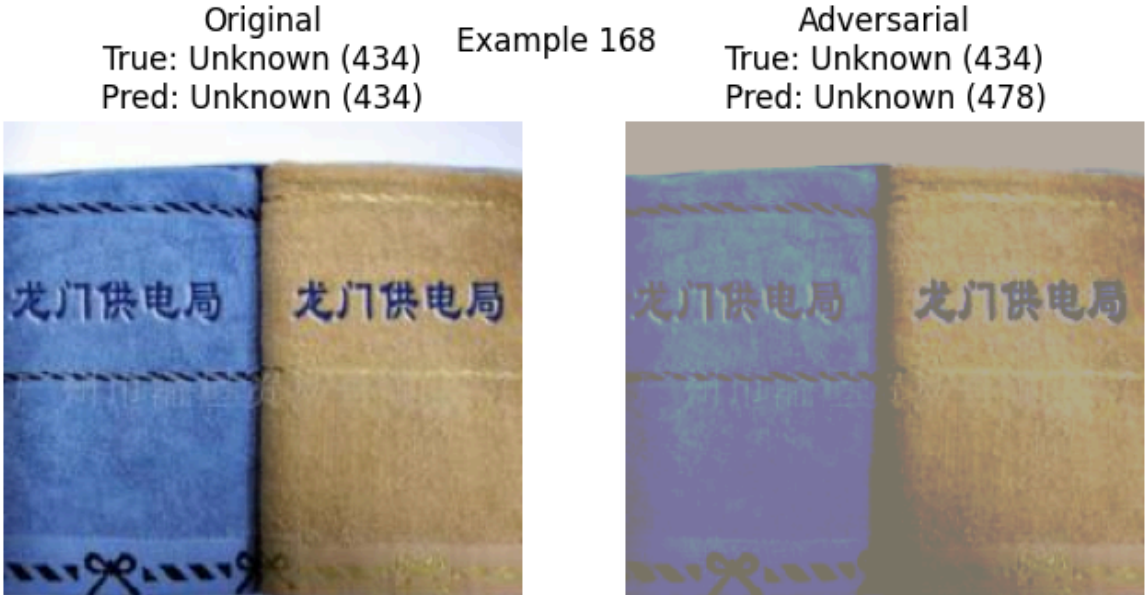
Visualization for Adversarial test set 1

```
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 (489)
Pred: Unknown (489)
Example 444
Adversarial
True: Unknown (489)
Pred: Unknown (489)

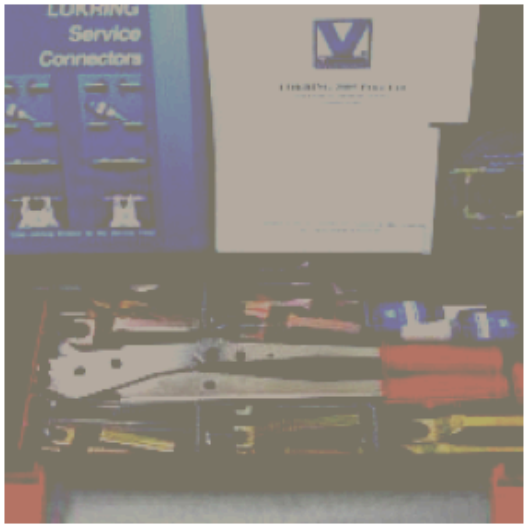

Original
True: Unknown (448)
Pred: Unknown (448)
Example 237
Adversarial
True: Unknown (448)
Pred: Unknown (448)


Original
True: Unknown (434)
Pred: Unknown (434)
Example 168
Adversarial
True: Unknown (434)
Pred: Unknown (478)
龙门供电局
龙门供电局
龙门供电局
龙门供电局

Original
True: Unknown (477)
Pred: Unknown (477)

Example 381

Adversarial
True: Unknown (477)
Pred: Unknown (477)



Original
True: Unknown (483)
Pred: Unknown (832)

Example 410

Adversarial
True: Unknown (483)
Pred: Unknown (403)

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, i
    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 [00:24<00:00,  1.53s/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, 18.38it/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)
```

Original
True: binder
Pred: binder

Example 229

Adversarial
True: binder
Pred: carton

Original
True: bow
Pred: bow

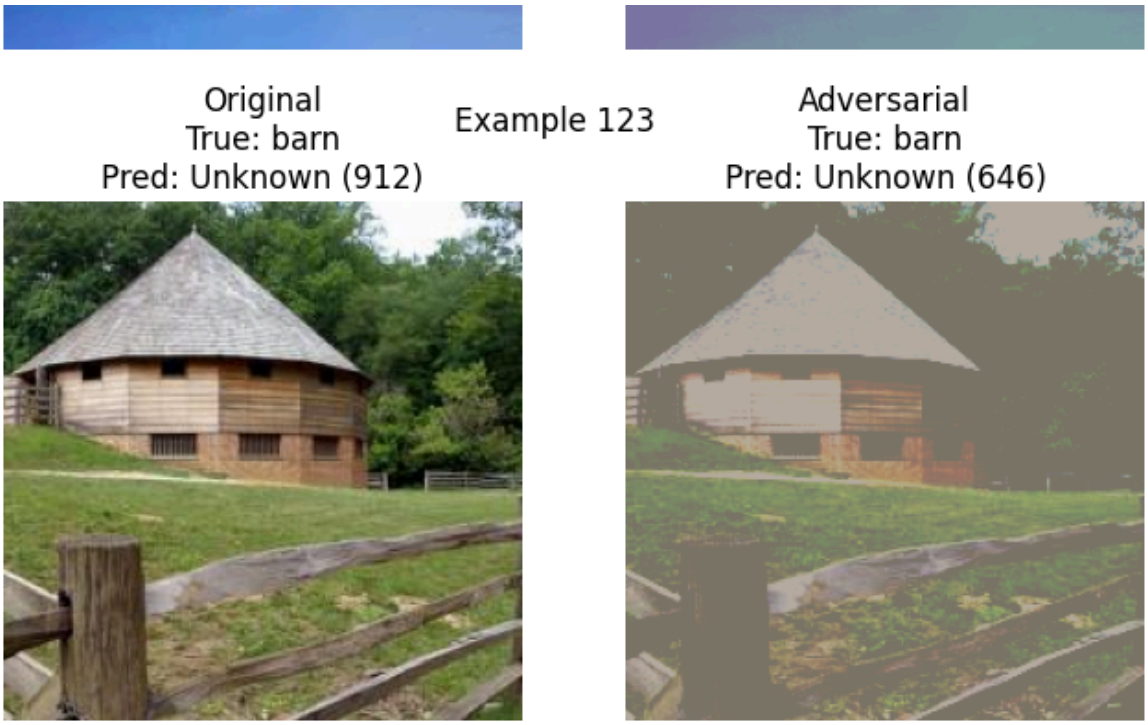
Example 277

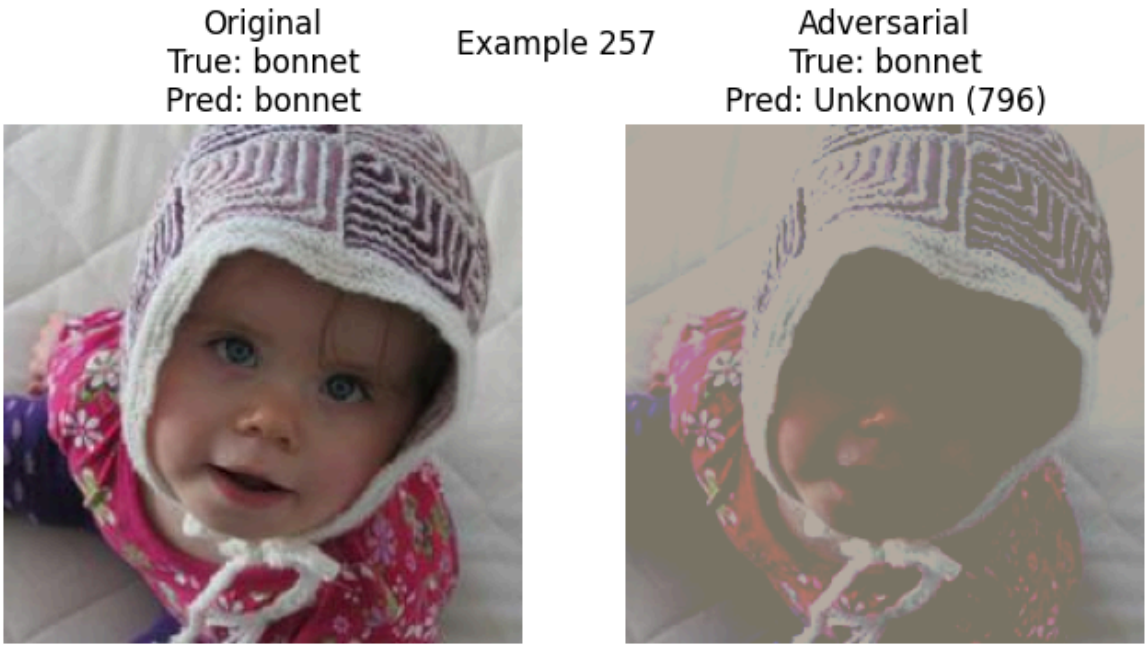
Adversarial
True: bow
Pred: Unknown (58)

Original
True: barbell
Pred: barbell

Example 109

Adversarial
True: barbell
Pred: Unknown (917)


Original
True: barn
Pred: Unknown (912)
Example 123
Adversarial
True: barn
Pred: Unknown (646)


Original
True: bonnet
Pred: bonnet
Example 257
Adversarial
True: bonnet
Pred: Unknown (796)

Task 4: Patch PGD Attack (fixed target)

```
def patch_pgd_attack(model, images, target_class, epsilon, alpha, iters, patch_size, x_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.

    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, :,
            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=image
        y_start = torch.randint(0, images.shape[2] - patch_size + 1, (batch_size,), device=image

        # 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,
        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,  3.07s/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=Fal

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, 18.00it/s]
Patch Attack Top-1 Accuracy: 37.80%
Patch Attack Top-5 Accuracy: 58.60%
```

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
    )
```

Example 216

Original
True: Unknown (444)
Pred: Unknown (444)

Adversarial (Patch)
True: Unknown (444)
Pred: Unknown (444)

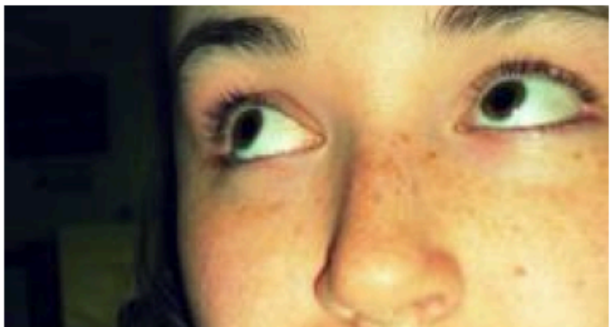
Example 131

Original
True: Unknown (427)
Pred: Unknown (653)

Adversarial (Patch)
True: Unknown (427)
Pred: Unknown (504)

Example 91

Original
True: Unknown (419)
Pred: Unknown (419)

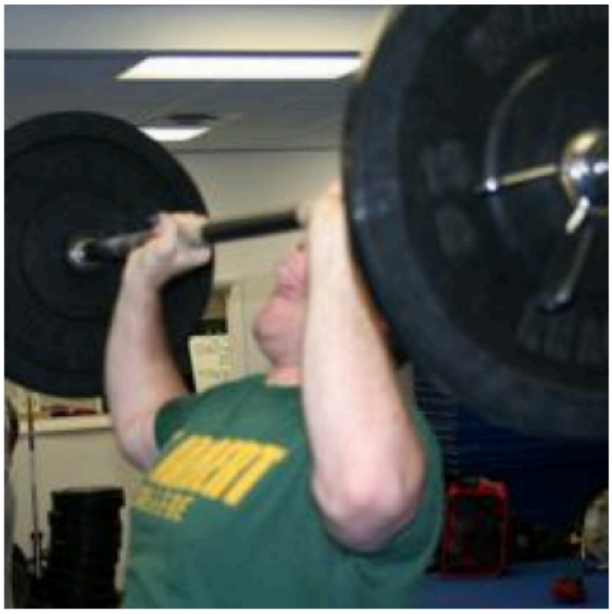


Adversarial (Patch)
True: Unknown (419)
Pred: Unknown (643)

Example 107

Original
True: Unknown (422)
Pred: Unknown (422)

Adversarial (Patch)
True: Unknown (422)
Pred: Unknown (845)



Example 368

Original
True: Unknown (474)
Pred: Unknown (474)

Adversarial (Patch)
True: Unknown (474)
Pred: Unknown (824)