

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: CxHxW tensors in [0,1] *after* unnormalization
    """
    # convert to HxWxC 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-b627a593.pth [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': 7, 'n02706490': 8, 'n02708143': 9, 'n02709418': 10, 'n02709586': 11, 'n02709820': 12, 'n02709956': 13, 'n02709963': 14, 'n02709964': 15, 'n02709965': 16, 'n02709966': 17, 'n02709967': 18, 'n02709968': 19, 'n02709969': 20, 'n02709970': 21, 'n02709971': 22, 'n02709972': 23, 'n02709973': 24, 'n02709974': 25, 'n02709975': 26, 'n02709976': 27, 'n02709977': 28, 'n02709978': 29, 'n02709979': 30, 'n02709980': 31, 'n02709981': 32, 'n02709982': 33, 'n02709983': 34, 'n02709984': 35, 'n02709985': 36, 'n02709986': 37, 'n02709987': 38, 'n02709988': 39, 'n02709989': 40, 'n02709990': 41, 'n02709991': 42, 'n02709992': 43, 'n02709993': 44, 'n02709994': 45, 'n02709995': 46, 'n02709996': 47, 'n02709997': 48, 'n02709998': 49, 'n02709999': 50, 'n02710196': 51, 'n02710207': 52, 'n02710208': 53, 'n02710209': 54, 'n02710210': 55, 'n02710211': 56, 'n02710212': 57, 'n02710213': 58, 'n02710214': 59, 'n02710215': 60, 'n02710216': 61, 'n02710217': 62, 'n02710218': 63, 'n02710219': 64, 'n02710220': 65, 'n02710221': 66, 'n02710222': 67, 'n02710223': 68, 'n02710224': 69, 'n02710225': 70, 'n02710226': 71, 'n02710227': 72, 'n02710228': 73, 'n02710229': 74, 'n02710230': 75, 'n02710231': 76, 'n02710232': 77, 'n02710233': 78, 'n02710234': 79, 'n02710235': 80, 'n02710236': 81, 'n02710237': 82, 'n02710238': 83, 'n02710239': 84, 'n02710240': 85, 'n02710241': 86, 'n02710242': 87, 'n02710243': 88, 'n02710244': 89, 'n02710245': 90, 'n02710246': 91, 'n02710247': 92, 'n02710248': 93, 'n02710249': 94, 'n02710250': 95, 'n02710251': 96, 'n02710252': 97, 'n02710253': 98, 'n02710254': 99, 'n02710255': 100, 'n02710256': 101, 'n02710257': 102, 'n02710258': 103, 'n02710259': 104, 'n02710260': 105, 'n02710261': 106, 'n02710262': 107, 'n02710263': 108, 'n02710264': 109, 'n02710265': 110, 'n02710266': 111, 'n02710267': 112, 'n02710268': 113, 'n02710269': 114, 'n02710270': 115, 'n02710271': 116, 'n02710272': 117, 'n02710273': 118, 'n02710274': 119, 'n02710275': 120, 'n02710276': 121, 'n02710277': 122, 'n02710278': 123, 'n02710279': 124, 'n02710280': 125, 'n02710281': 126, 'n02710282': 127, 'n02710283': 128, 'n02710284': 129, 'n02710285': 130, 'n02710286': 131, 'n02710287': 132, 'n02710288': 133, 'n02710289': 134, 'n02710290': 135, 'n02710291': 136, 'n02710292': 137, 'n02710293': 138, 'n02710294': 139, 'n02710295': 140, 'n02710296': 141, 'n02710297': 142, 'n02710298': 143, 'n02710299': 144, 'n02710300': 145, 'n02710301': 146, 'n02710302': 147, 'n02710303': 148, 'n02710304': 149, 'n02710305': 150, 'n02710306': 151, 'n02710307': 152, 'n02710308': 153, 'n02710309': 154, 'n02710310': 155, 'n02710311': 156, 'n02710312': 157, 'n02710313': 158, 'n02710314': 159, 'n02710315': 160, 'n02710316': 161, 'n02710317': 162, 'n02710318': 163, 'n02710319': 164, 'n02710320': 165, 'n02710321': 166, 'n02710322': 167, 'n02710323': 168, 'n02710324': 169, 'n02710325': 170, 'n02710326': 171, 'n02710327': 172, 'n02710328': 173, 'n02710329': 174, 'n02710330': 175, 'n02710331': 176, 'n02710332': 177, 'n02710333': 178, 'n02710334': 179, 'n02710335': 180, 'n02710336': 181, 'n02710337': 182, 'n02710338': 183, 'n02710339': 184, 'n02710340': 185, 'n02710341': 186, 'n02710342': 187, 'n02710343': 188, 'n02710344': 189, 'n02710345': 190, 'n02710346': 191, 'n02710347': 192, 'n02710348': 193, 'n02710349': 194, 'n02710350': 195, 'n02710351': 196, 'n02710352': 197, 'n02710353': 198, 'n02710354': 199, 'n02710355': 200, 'n02710356': 201, 'n02710357': 202, 'n02710358': 203, 'n02710359': 204, 'n02710360': 205, 'n02710361': 206, 'n02710362': 207, 'n02710363': 208, 'n02710364': 209, 'n02710365': 210, 'n02710366': 211, 'n02710367': 212, 'n02710368': 213, 'n02710369': 214, 'n02710370': 215, 'n02710371': 216, 'n02710372': 217, 'n02710373': 218, 'n02710374': 219, 'n02710375': 220, 'n02710376': 221, 'n02710377': 222, 'n02710378': 223, 'n02710379': 224, 'n02710380': 225, 'n02710381': 226, 'n02710382': 227, 'n02710383': 228, 'n02710384': 229, 'n02710385': 230, 'n02710386': 231, 'n02710387': 232, 'n02710388': 233, 'n02710389': 234, 'n02710390': 235, 'n02710391': 236, 'n02710392': 237, 'n02710393': 238, 'n02710394': 239, 'n02710395': 240, 'n02710396': 241, 'n02710397': 242, 'n02710398': 243, 'n02710399': 244, 'n02710400': 245, 'n02710401': 246, 'n02710402': 247, 'n02710403': 248, 'n02710404': 249, 'n02710405': 250, 'n02710406': 251, 'n02710407': 252, 'n02710408': 253, 'n02710409': 254, 'n02710410': 255, 'n02710411': 256, 'n02710412': 257, 'n02710413': 258, 'n02710414': 259, 'n02710415': 260, 'n02710416': 261, 'n02710417': 262, 'n02710418': 263, 'n02710419': 264, 'n02710420': 265, 'n02710421': 266, 'n02710422': 267, 'n02710423': 268, 'n02710424': 269, 'n02710425': 270, 'n02710426': 271, 'n02710427': 272, 'n02710428': 273, 'n02710429': 274, 'n02710430': 275, 'n02710431': 276, 'n02710432': 277, 'n02710433': 278, 'n02710434': 279, 'n02710435': 280, 'n02710436': 281, 'n02710437': 282, 'n02710438': 283, 'n02710439': 284, 'n02710440': 285, 'n02710441': 286, 'n02710442': 287, 'n02710443': 288, 'n02710444': 289, 'n02710445': 290, 'n02710446': 291, 'n02710447': 292, 'n02710448': 293, 'n02710449': 294, 'n02710450': 295, 'n02710451': 296, 'n02710452': 297, 'n02710453': 298, 'n02710454': 299, 'n02710455': 300, 'n02710456': 301, 'n02710457': 302, 'n02710458': 303, 'n02710459': 304, 'n02710460': 305, 'n02710461': 306, 'n02710462': 307, 'n02710463': 308, 'n02710464': 309, 'n02710465': 310, 'n02710466': 311, 'n02710467': 312, 'n02710468': 313, 'n02710469': 314, 'n02710470': 315, 'n02710471': 316, 'n02710472': 317, 'n02710473': 318, 'n02710474': 319, 'n02710475': 320, 'n02710476': 321, 'n02710477': 322, 'n02710478': 323, 'n02710479': 324, 'n02710480': 325, 'n02710481': 326, 'n02710482': 327, 'n02710483': 328, 'n02710484': 329, 'n02710485': 330, 'n02710486': 331, 'n02710487': 332, 'n02710488': 333, 'n02710489': 334, 'n02710490': 335, 'n02710491': 336, 'n02710492': 337, 'n02710493': 338, 'n02710494': 339, 'n02710495': 340, 'n02710496': 341, 'n02710497': 342, 'n02710498': 343, 'n02710499': 344, 'n02710500': 345, 'n02710501': 346, 'n02710502': 347, 'n02710503': 348, 'n02710504': 349, 'n02710505': 350, 'n02710506': 351, 'n02710507': 352, 'n02710508': 353, 'n02710509': 354, 'n02710510': 355, 'n02710511': 356, 'n02710512': 357, 'n02710513': 358, 'n02710514': 359, 'n02710515': 360, 'n02710516': 361, 'n02710517': 362, 'n02710518': 363, 'n02710519': 364, 'n02710520': 365, 'n02710521': 366, 'n02710522': 367, 'n02710523': 368, 'n02710524': 369, 'n02710525': 370, 'n02710526': 371, 'n02710527': 372, 'n02710528': 373, 'n02710529': 374, 'n02710530': 375, 'n02710531': 376, 'n02710532': 377, 'n02710533': 378, 'n02710534': 379, 'n02710535': 380, 'n02710536': 381, 'n02710537': 382, 'n02710538': 383, 'n02710539': 384, 'n02710540': 385, 'n02710541': 386, 'n02710542': 387, 'n02710543': 388, 'n02710544': 389, 'n02710545': 390, 'n02710546': 391, 'n02710547': 392, 'n02710548': 393, 'n02710549': 394, 'n02710550': 395, 'n02710551': 396, 'n02710552': 397, 'n02710553': 398, 'n02710554': 399, 'n02710555': 400, 'n02710556': 401, 'n02710557': 402, 'n02710558': 403, 'n02710559': 404, 'n02710560': 405, 'n02710561': 406, 'n02710562': 407, 'n02710563': 408, 'n02710564': 409, 'n02710565': 410, 'n02710566': 411, 'n02710567': 412, 'n02710568': 413, 'n02710569': 414, 'n02710570': 415, 'n02710571': 416, 'n02710572': 417, 'n02710573': 418, 'n02710574': 419, 'n02710575': 420, 'n02710576': 421, 'n02710577': 422, 'n02710578': 423, 'n02710579': 424, 'n02710580': 425, 'n02710581': 426, 'n02710582': 427, 'n02710583': 428, 'n02710584': 429, 'n02710585': 430, 'n02710586': 431, 'n02710587': 432, 'n02710588': 433, 'n02710589': 434, 'n02710590': 435, 'n02710591': 436, 'n02710592': 437, 'n02710593': 438, 'n02710594': 439, 'n02710595': 440, 'n02710596': 441, 'n02710597': 442, 'n02710598': 443, 'n02710599': 444, 'n02710600': 445, 'n02710601': 446, 'n02710602': 447, 'n02710603': 448, 'n02710604': 449, 'n02710605': 450, 'n02710606': 451, 'n02710607': 452, 'n02710608': 453, 'n02710609': 454, 'n02710610': 455, 'n02710611': 456, 'n02710612': 457, 'n02710613': 458, 'n02710614': 459, 'n02710615': 460, 'n02710616': 461, 'n02710617': 462, 'n02710618': 463, 'n02710619': 464, 'n02710620': 465, 'n02710621': 466, 'n02710622': 467, 'n02710623': 468, 'n02710624': 469, 'n02710625': 470, 'n02710626': 471, 'n02710627': 472, 'n02710628': 473, 'n02710629': 474, 'n02710630': 475, 'n02710631': 476, 'n02710632': 477, 'n02710633': 478, 'n02710634': 479, 'n02710635': 480, 'n02710636': 481, 'n02710637': 482, 'n02710638': 483, 'n02710639': 484, 'n02710640': 485, 'n02710641': 486, 'n02710642': 487, 'n02710643': 488, 'n02710644': 489, 'n02710645': 490, 'n02710646': 491, 'n02710647': 492, 'n02710648': 493, 'n02710649': 494, 'n02710650': 495, 'n02710651': 496, 'n02710652': 497, 'n02710653': 498, 'n02710654': 499, 'n02710655': 500, 'n02710656': 501, 'n02710657': 502, 'n02710658': 503, 'n02710659': 504, 'n02710660': 505, 'n02710661': 506, 'n02710662': 507, 'n02710663': 508, 'n02710664': 509, 'n02710665': 510, 'n02710666': 511, 'n02710667': 512, 'n02710668': 513, 'n02710669': 514, 'n02710670': 515, 'n02710671': 516, 'n02710672': 517, 'n02710673': 518, 'n02710674': 519, 'n02710675': 520, 'n02710676': 521, 'n02710677': 522, 'n02710678': 523, 'n02710679': 524, 'n02710680': 525, 'n02710681': 526, 'n02710682': 527, 'n02710683': 528, 'n02710684': 529, 'n02710685': 530, 'n02710686': 531, 'n02710687': 532, 'n02710688': 533, 'n02710689': 534, 'n02710690': 535, 'n02710691': 536, 'n02710692': 537, 'n02710693': 538, 'n02710694': 539, 'n02710695': 540, 'n02710696': 541, 'n02710697': 542, 'n02710698': 543, 'n02710699': 544, 'n02710700': 545, 'n02710701': 546, 'n02710702': 547, 'n02710703': 548, 'n02710704': 549, 'n02710705': 550, 'n02710706': 551, 'n02710707': 552, 'n02710708': 553, 'n02710709': 554, 'n02710710': 555, 'n02710711': 556, 'n02710712': 557, 'n02710713': 558, 'n02710714': 559, 'n02710715': 560, 'n02710716': 561, 'n02710717': 562, 'n02710718': 563, 'n02710719': 564, 'n02710720': 565, 'n02710721': 566, 'n02710722': 567, 'n02710723': 568, 'n02710724': 569, 'n02710725': 570, 'n02710726': 571, 'n02710727': 572, 'n02710728': 573, 'n02710729': 574, 'n02710730': 575, 'n02710731': 576, 'n02710732': 577, 'n02710733': 578, 'n02710734': 579, 'n02710735': 580, 'n02710736': 581, 'n02710737': 582, 'n02710738': 583, 'n02710739': 584, 'n02710740': 585, 'n02710741': 586, 'n02710742': 587, 'n02710743': 588, 'n02710744': 589, 'n02710745': 590, 'n02710746': 591, 'n02710747': 592, 'n02710748': 593, 'n02710749': 594, 'n02710750': 595, 'n02710751': 596, 'n02710752': 597, 'n02710753': 598, 'n02710754': 599, 'n02710755': 600, 'n02710756': 601, 'n02710757': 602, 'n02710758': 603, 'n02710759': 604, 'n02710760': 605, 'n02710761': 606, 'n02710762': 607, 'n02710763': 608, 'n02710764': 609, 'n02710765': 610, 'n02710766': 611, 'n02710767': 612, 'n02710768': 613, 'n02710769': 614, 'n02710770': 615, 'n02710771': 616, 'n02710772': 617, 'n02710773': 618, 'n02710774': 619, 'n02710775': 620, 'n02710776': 621, 'n02710777': 622, 'n02710778': 623, 'n02710779': 624, 'n02710780': 625, 'n02710781': 626, 'n02710782': 627, 'n02710783': 628, 'n02710784': 629, 'n02710785': 630, 'n02710786': 631, 'n02710787': 632, 'n02710788': 633, 'n02710789': 634, 'n02710790': 635, 'n02710791': 636, 'n02710792': 637, 'n02710793': 638, 'n02710794': 639, 'n02710795': 640, 'n02710796': 641, 'n02710797': 642, 'n02710798': 643, 'n02710799': 644, 'n02710800': 645, 'n02710801': 646, 'n02710802': 647, 'n02710803': 648, 'n02710804': 649, 'n02710805': 650, 'n02710806': 651, 'n02710807': 652, 'n02710808': 653, 'n02710809': 654, 'n02710810': 655, 'n02710811': 656, 'n02710812': 657, 'n02710813': 658, 'n02710814': 659, 'n02710815': 660, 'n02710816': 661, 'n02710817': 662, 'n02710818': 663, 'n02710819': 664, 'n02710820': 665, 'n02710821': 666, 'n02710822': 667, 'n02710823': 668, 'n02710824': 669, 'n02710825': 670, 'n02710826': 671, 'n02710827': 672, 'n02710828': 673, 'n02710829': 674, 'n02710830': 675, 'n02710831': 676, 'n02710832': 677, 'n02710833': 678, 'n02710834': 679, 'n02710835': 680, 'n02710836': 681, 'n02710837': 682, 'n02710838': 683, 'n02710839': 684, 'n02710840': 685, 'n02710841': 686, 'n02710842': 687, 'n02710843': 688, 'n02710844': 689, 'n02710845': 690, 'n02710846': 691, 'n02710847': 692, 'n02710848': 693, 'n02710849': 694, 'n02710850': 695, 'n02710851': 696, 'n02710852': 697, 'n02710853': 698, 'n02710854': 699, 'n02710855': 700, 'n02710856': 701, 'n02710857': 702, 'n02710858': 703, 'n02710859': 704, 'n02710860': 705, 'n02710861': 706, 'n02710862': 707, 'n02710863': 708, 'n02710864': 709, 'n02710865': 710, 'n02710866': 711, 'n02710867': 712, 'n02710868': 713, 'n02710869': 714, 'n02710870': 715, 'n02710871': 716, 'n02710872': 717, 'n02710873': 718, 'n02710874': 719

```
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)

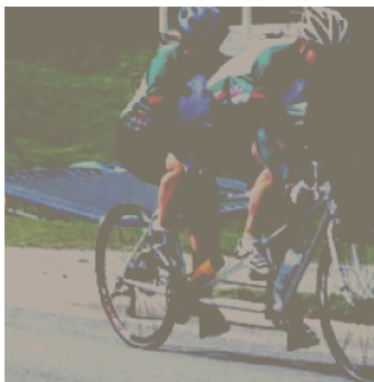
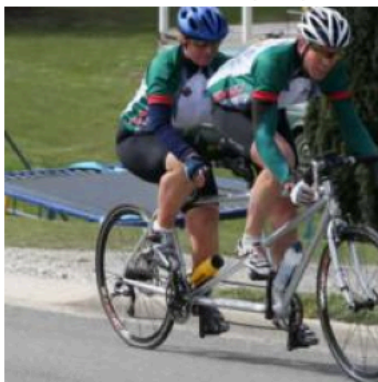
```



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

Example 216

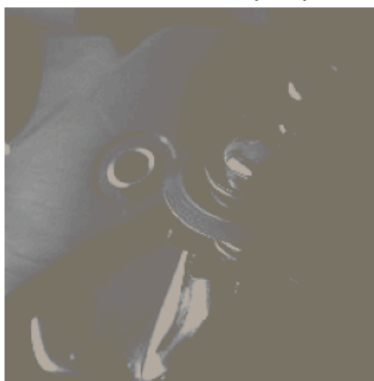
Adversarial
True: Unknown (444)
Pred: Unknown (671)



Original
True: Unknown (473)
Pred: Unknown (473)

Example 363

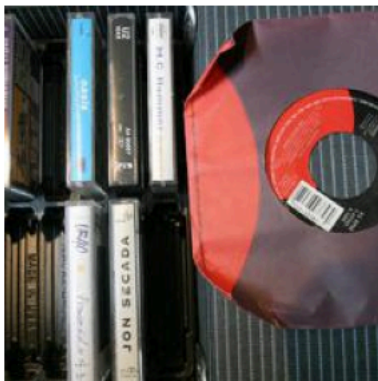
Adversarial
True: Unknown (473)
Pred: Unknown (999)



Original
True: Unknown (482)
Pred: Unknown (453)

Example 408

Adversarial
True: Unknown (482)
Pred: Unknown (769)

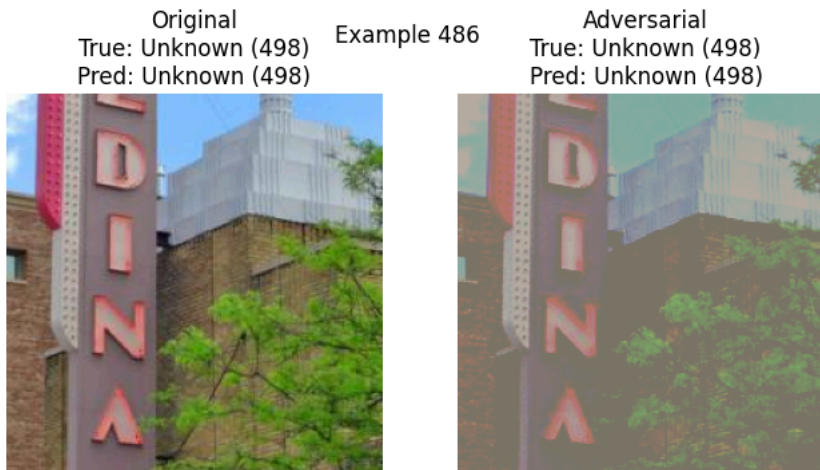


Original
True: Unknown (468)
Pred: Unknown (436)

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 $\epsilon \rightarrow$ | Stronger attacks but riskier visually |

| Increase $\alpha \rightarrow$ | Faster movement, but needs to be balanced |

| Increase iters \rightarrow | 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)
```




Original
True: chain mail
Pred: chain mail



Example 446

Adversarial
True: chain mail
Pred: Unknown (399)

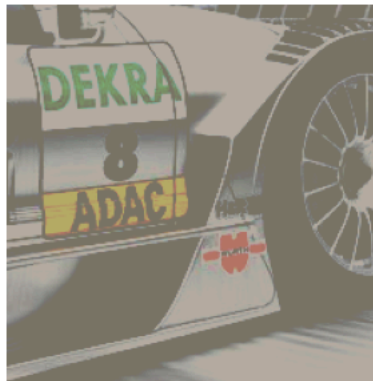


Original
True: car wheel
Pred: Unknown (751)



Example 390

Adversarial
True: car wheel
Pred: Unknown (893)



Original
True: caldron
Pred: caldron



Example 343

Adversarial
True: caldron
Pred: Unknown (517)



✓ 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]
```



```

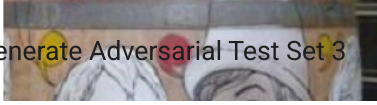
perturbed_images[j, :, ys:ys+patch_size, xs:xs+patch_size] += alpha * grad[j, :, ys:ys+patch_size, xs:xs+patch_size]
eta = torch.clamp(perturbed_images[j] - ori_images[j], min=-epsilon, max=epsilon)
perturbed_images[j] = torch.clamp(ori_images[j] + eta, min=0, max=1)

```

```
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
    )

```

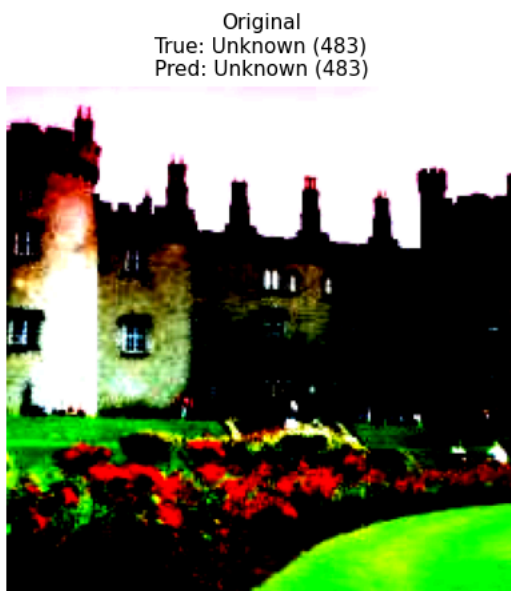
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for

Example 317



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



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

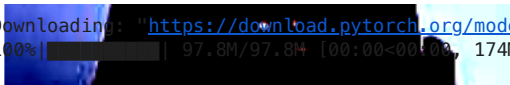
Example 302

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.pth> 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')

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

```

```

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)

```

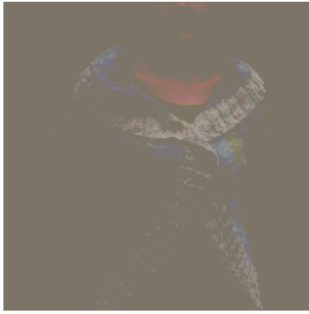
```

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

```

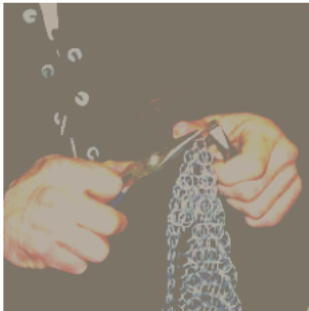
True: Unknown (474)

Pred: Unknown (501)



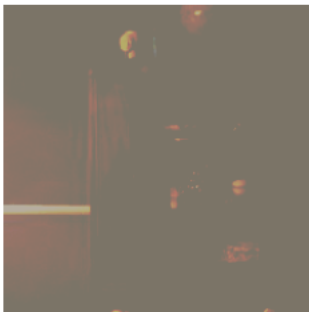
True: Unknown (490)

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)

