Table of contents

Imports

In [2]:

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch
import torchvision
import einops
import random
import wandb

from torch import Tensor
from matplotlib import cm
from icecream import ic
from typing import Union, List, Tuple, Literal
from tqdm.auto import tqdm, trange
from IPython.display import display, Markdown
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset, Dataset
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, Conf
from skimage.feature import hog
from skimage import exposure
```

In [ ]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_available() e
print(f"using {device.type}")
```

```
using mps
```

Loading the model

In [4]:

```
TRAIN_DATA_PATH = 'data/network visualization'
MODEL_WEIGHTS = 'network_visualization.pth'
MEAN = np.array([0.485, 0.456, 0.406])
STD = np.array([0.229, 0.224, 0.225])
CLASSES = [
    "arctic fox", "corgi", "electric ray", "goldfish", "hammerhead shark", "horse", "hummingbird", "indigo
]
```

In [ ]:

```
model = torchvision.models.resnet18()
model.fc = nn.Linear(in_features=512, out_features=10, bias=True)
model.load_state_dict(torch.load("./network_visualization.pth", map_location=device))
model.to(device)
```

Out[ ]:

```
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)
```

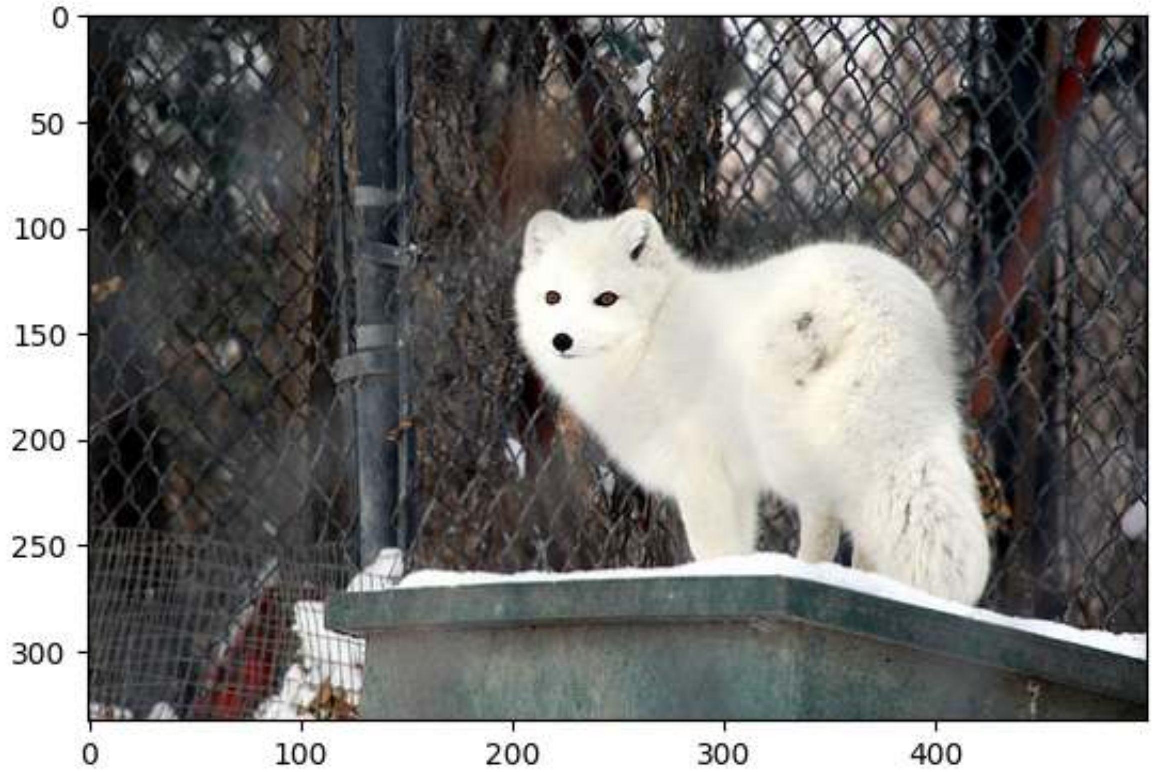
```
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1,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

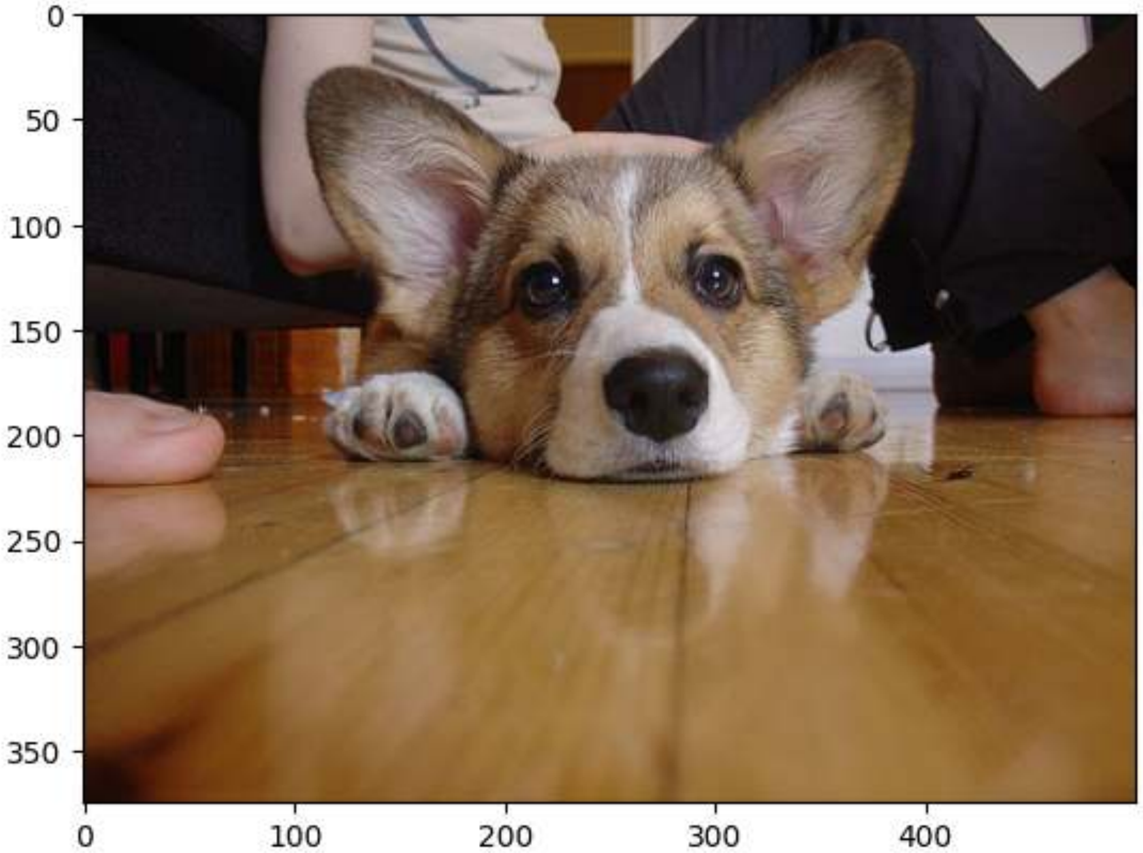
```
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (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)
    )
  )
  (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)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

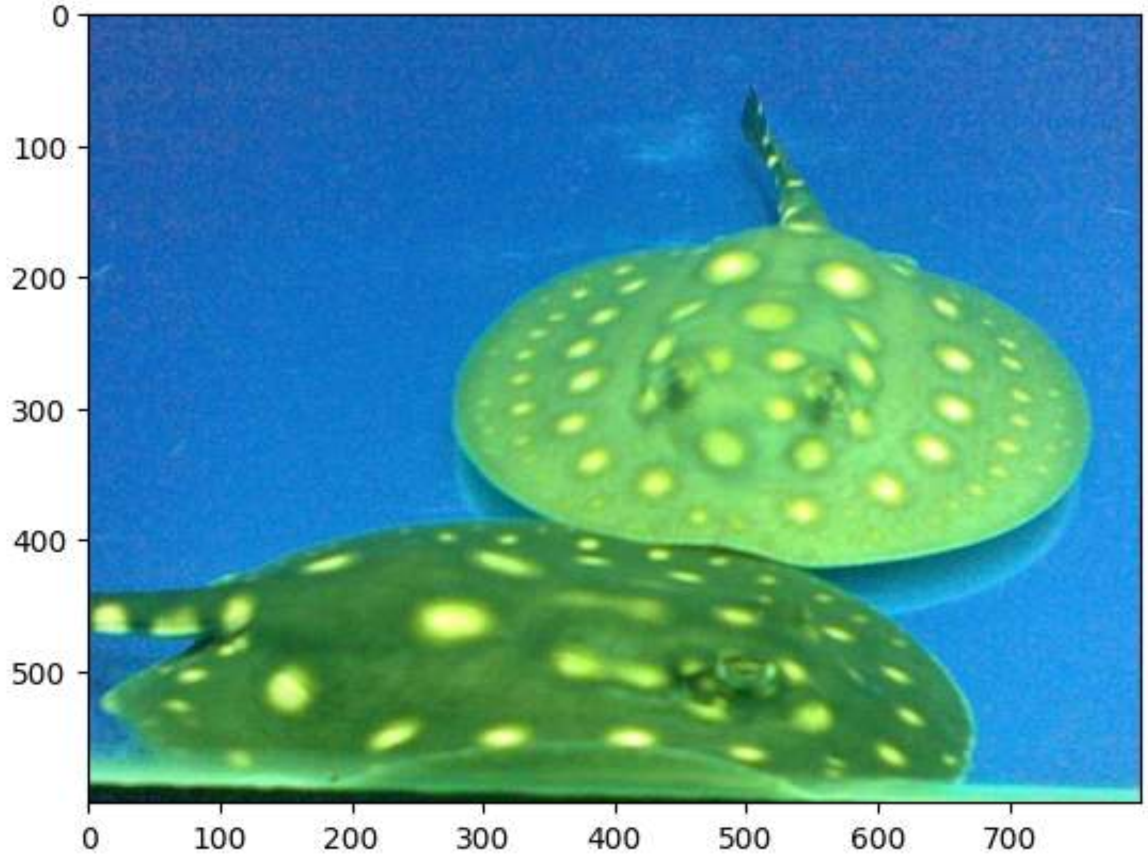
Constructing the dataset

In [6]:
```
img_indices = [random.randint(1, 5) for _ in range(len(CLASSES))]
img_paths = [f"./data/network visualization/{CLASSES[idx]}/{CLASSES[idx]}_{img_num}.JPEG" for idx, img_num
```

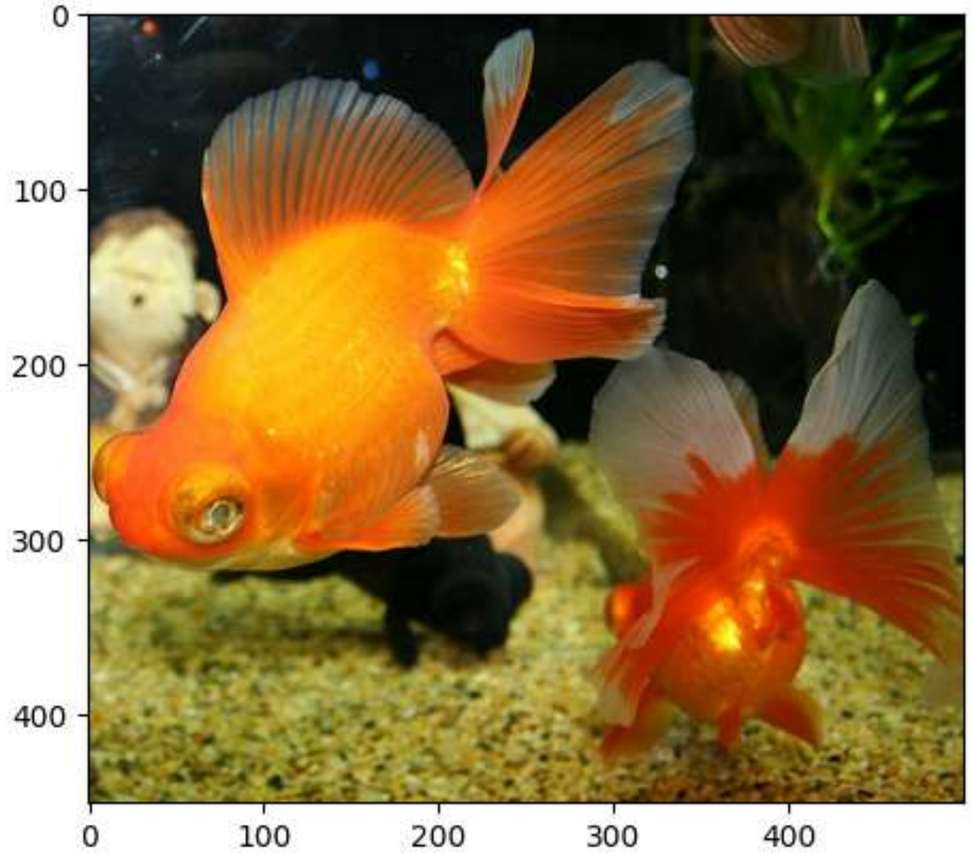
In [7]:
```
def visualize_raw_images(img_paths: list):
    for img_path in img_paths:
        plt.imshow(cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB))
        plt.show()

visualize_raw_images(img_paths)
```




0
50
100
150
200
250
300
350
0
100
200
300
400

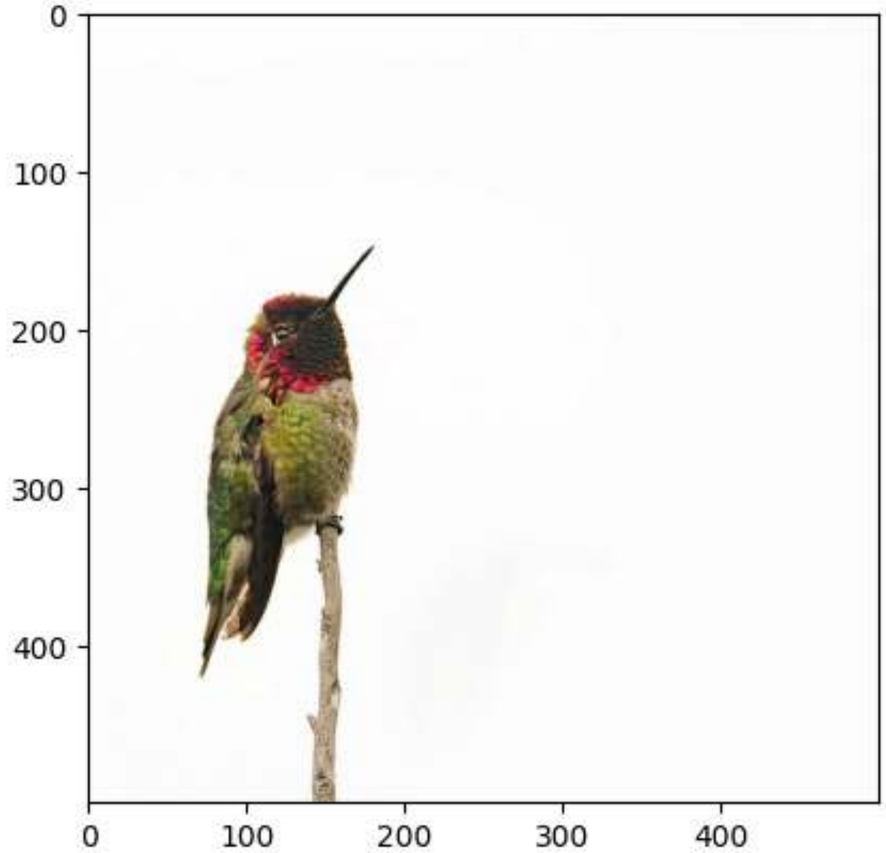


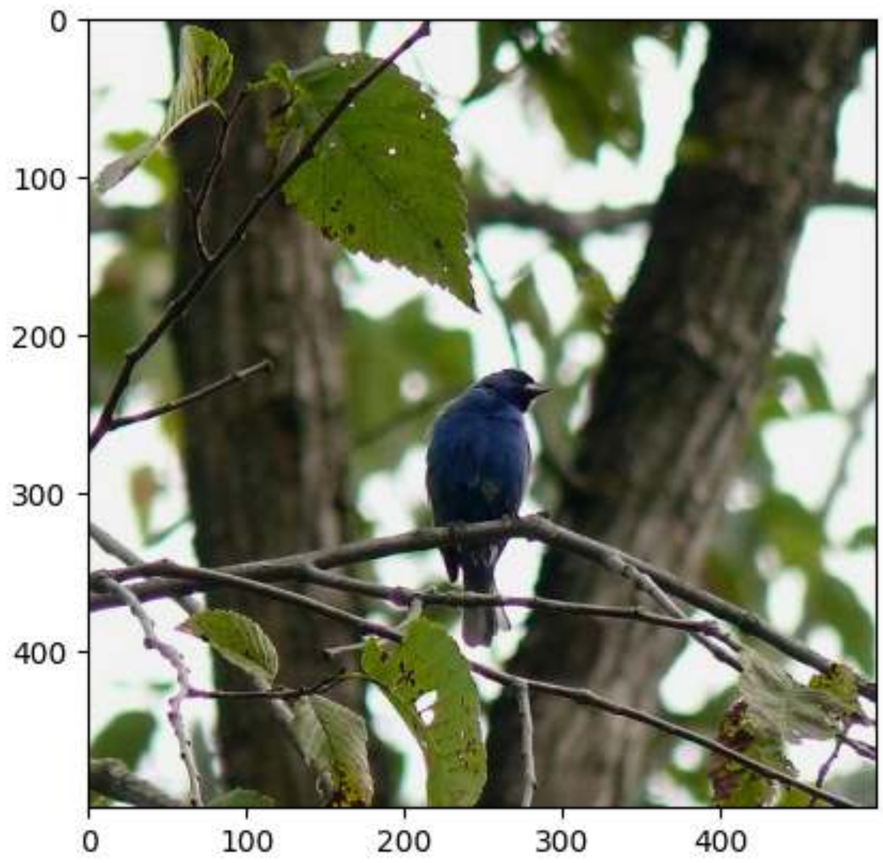
0
100
200
300
400
500
0
100
200
300
400
500
600
700

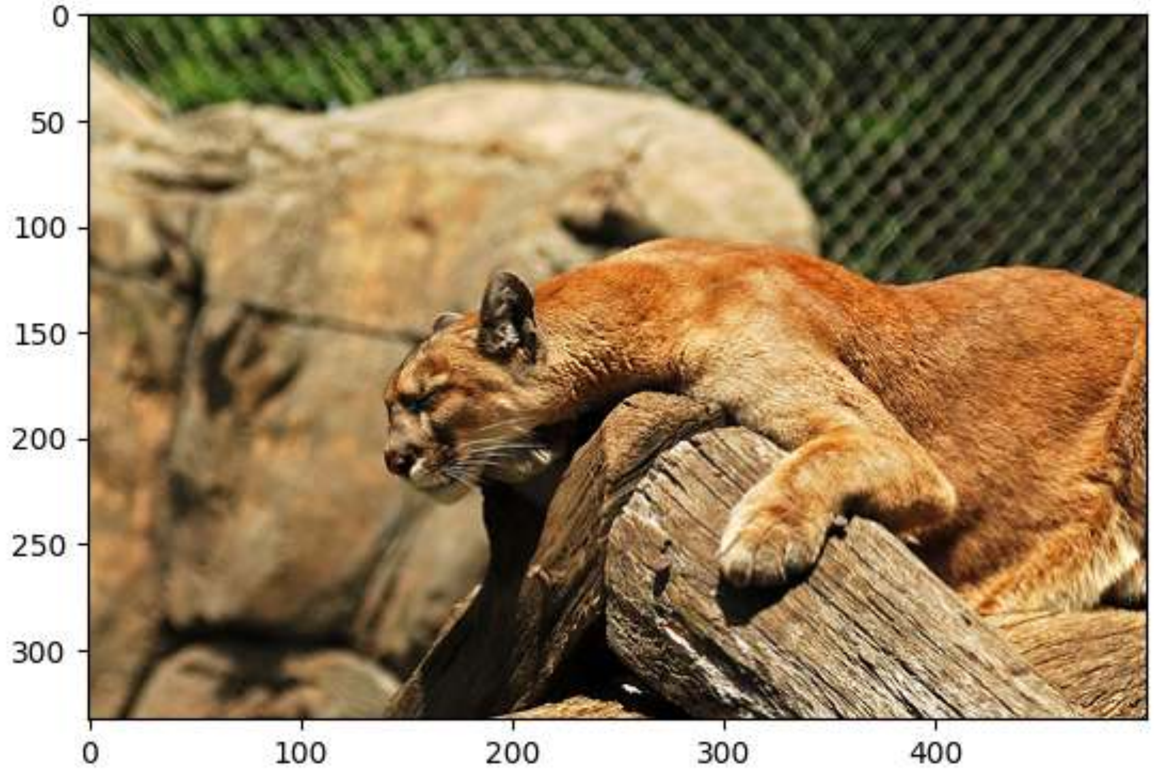

0
100
200
300
400
0
100
200
300
400

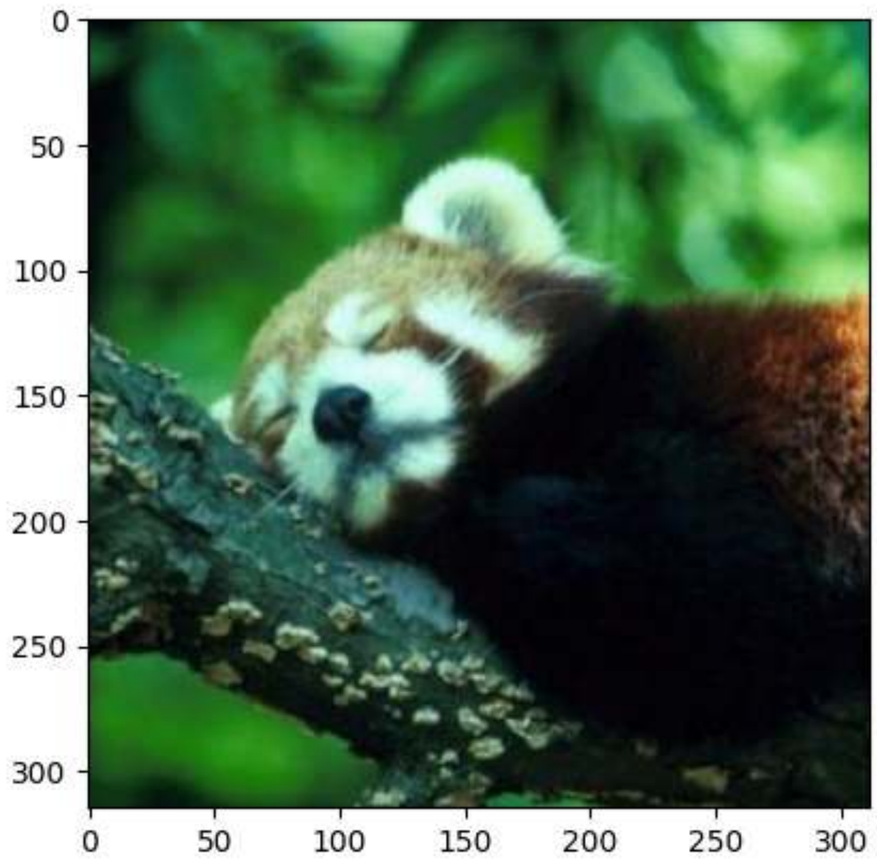

0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
100
200
300
400


0
100
200
300
400
0
100
200
300
400


0
100
200
300
400
0
100
200
300
400


0
50
100
150
200
250
300
0
100
200
300
400



In [8]:
```
class CuteDataset(Dataset):
    def __init__(self, img_paths: list, labels: list, transform=None):
        super().__init__()
        self.data = [cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB) for img_path in img_paths]
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.data[idx]
```

```python
        label = self.labels[idx]
        if self.transform:
            sample = self.transform(sample)
        return sample, label
```

In [9]:
```python
def preprocess_data(img_paths: list, labels: list, mean: list=MEAN, std: list=STD):
    transformations = torchvision.transforms.Compose([
        torchvision.transforms.ToPILImage(),
        torchvision.transforms.Resize((224, 224)),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean=mean, std=std),
    ])
    dataset = CuteDataset(img_paths, labels, transformations)
    return dataset
```

In [10]:
```python
cute_dataset = preprocess_data(img_paths, [i for i in range(len(CLASSES))], MEAN, STD)
```

In [ ]:
```python
len(cute_dataset)
```

Out[ ]:
```
10
```

Adding small amount of Gaussian noise and calculating accuracy

In [12]:
```python
def add_gauss_noise_to_img(img, mean, std, eps):
    img = einops.rearrange(img, 'c h w -> h w c')
    noise = torch.randn_like(img) * std + mean
    res = img + eps * noise
    return res.to(torch.float32)
```

In [13]:
```python
def min_max_scale_img(img: Tensor):
    img = (img - img.min()) / (img.max() - img.min()) * 255
    # if tensor, convert to numpy
    if isinstance(img, Tensor):
```

```
        img = img.cpu().numpy()
    img = img.astype(np.uint8)
    return img
```

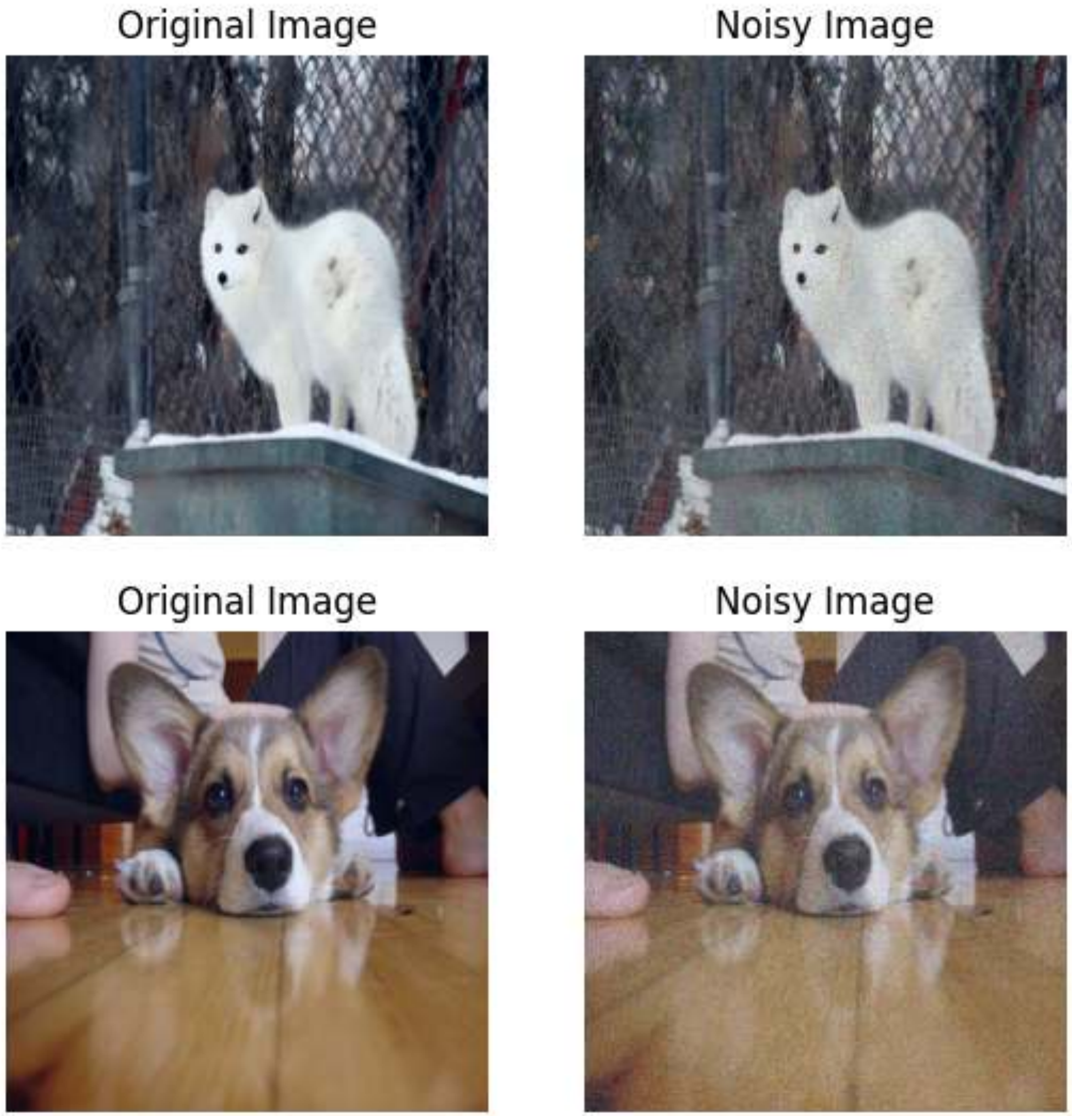
In [14]:

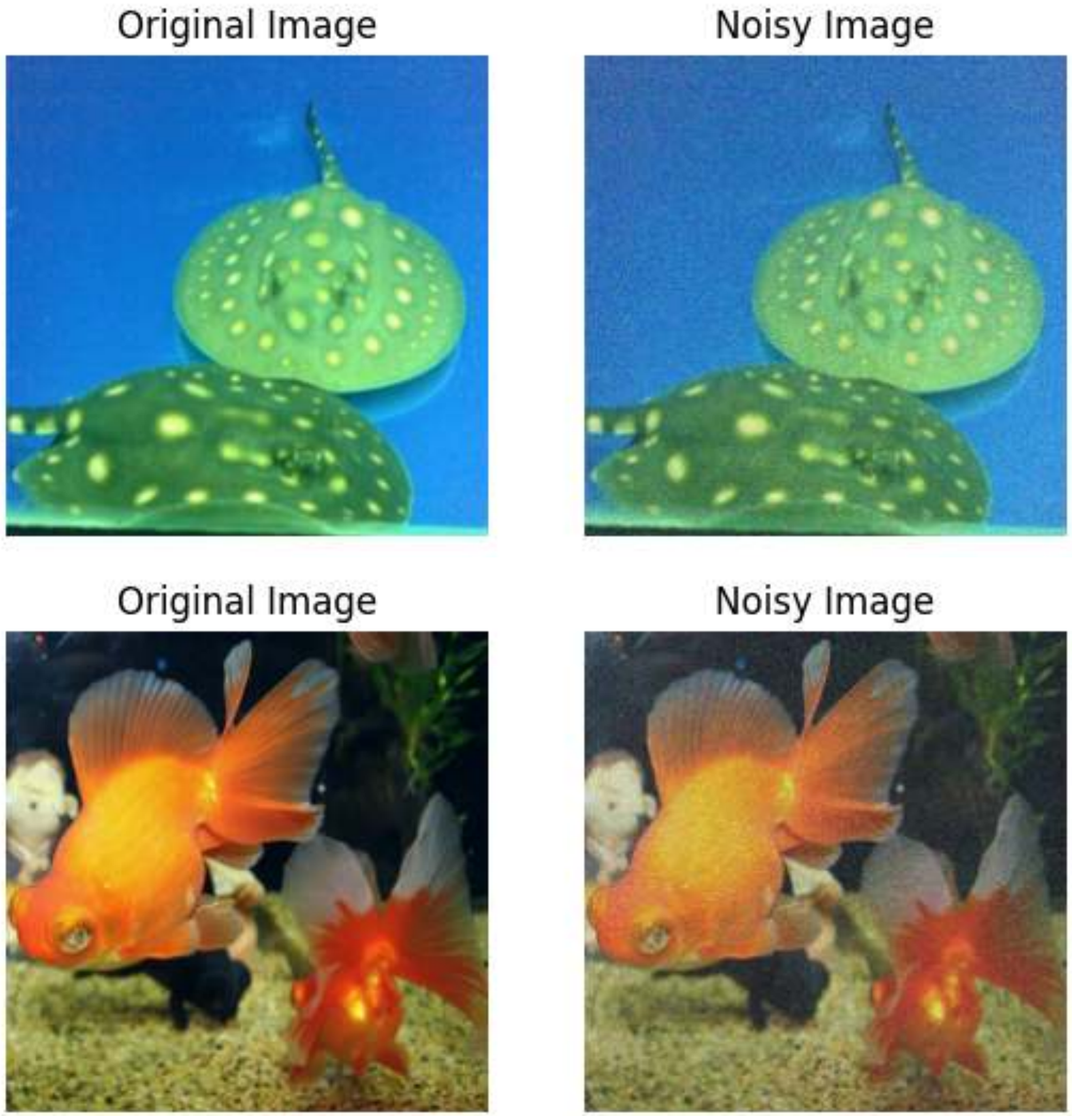
```
def accuracy_on_gauss_images(model: nn.Module, dataset: Dataset, mean: list=MEAN, std: list=STD, eps:float=
    model.eval()
    dataloader = DataLoader(dataset, batch_size=1, shuffle=False)
    correct = 0

    for img, label in dataloader:
        fig, axs = plt.subplots(1, 2)
        axs[0].imshow(min_max_scale_img(einops.rearrange(img, '1 c h w -> h w c')))
        axs[0].set_title("Original Image")
        axs[0].axis("off")
        img = add_gauss_noise_to_img(img[0], mean, std, eps).to(device)
        axs[1].imshow(min_max_scale_img(img))
        axs[1].set_title("Noisy Image")
        axs[1].axis("off")
        label = label.to(device)
        with torch.no_grad():
            img = einops.rearrange(img, 'h w c -> 1 c h w')
            output = model(img.to(torch.float32))
            pred = nn.Softmax(dim=1)(output)
            pred = torch.argmax(pred, dim=1)
            correct += pred.eq(label).sum().item()

    return correct / len(dataset)
```

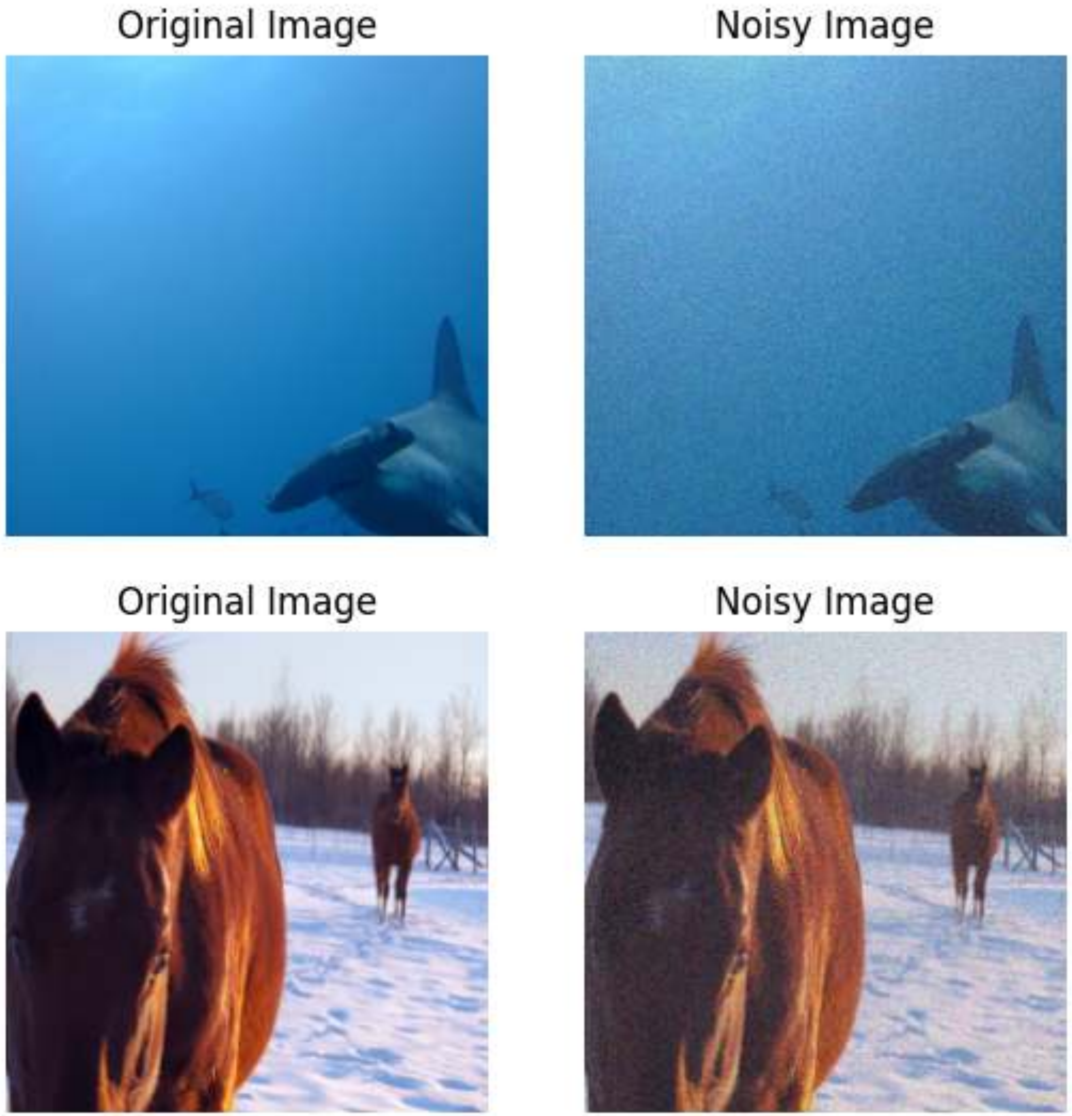
In [15]:

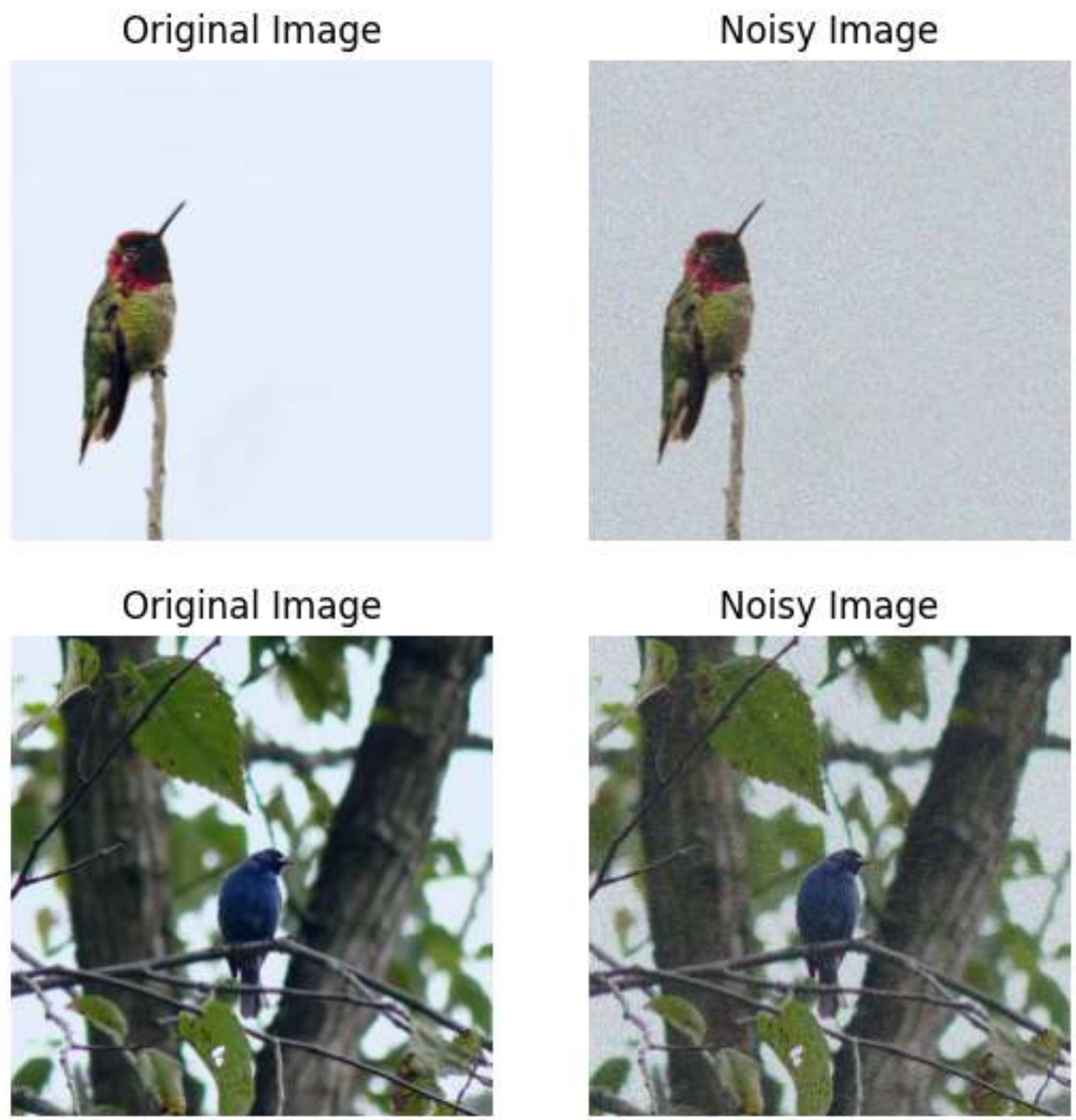
```
acc = accuracy_on_gauss_images(model, cute_dataset, MEAN, STD, 1)
print(f"Accuracy on gaussian noise added images: {acc}")
```

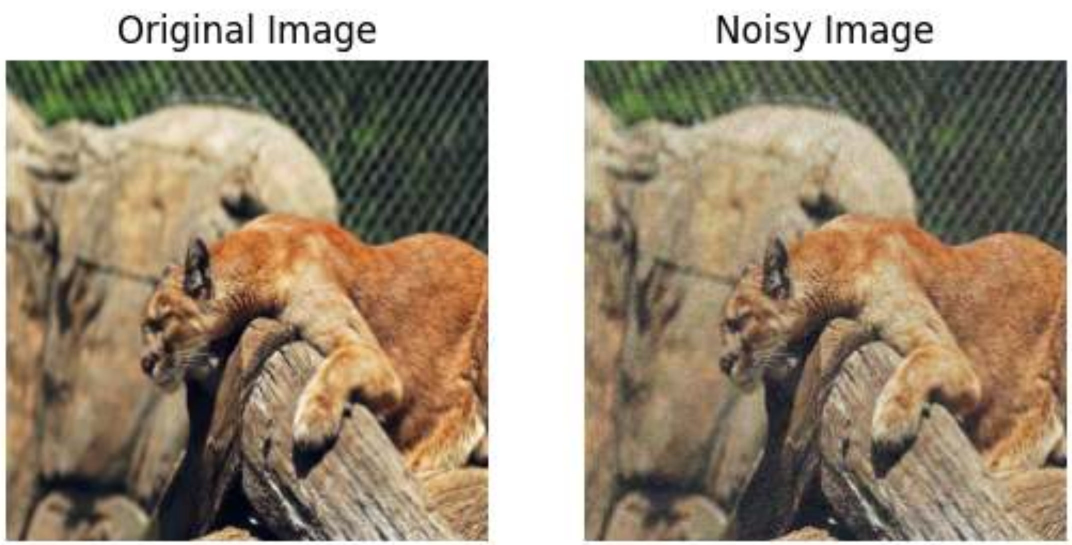
```
Accuracy on gaussian noise added images: 1.0
```

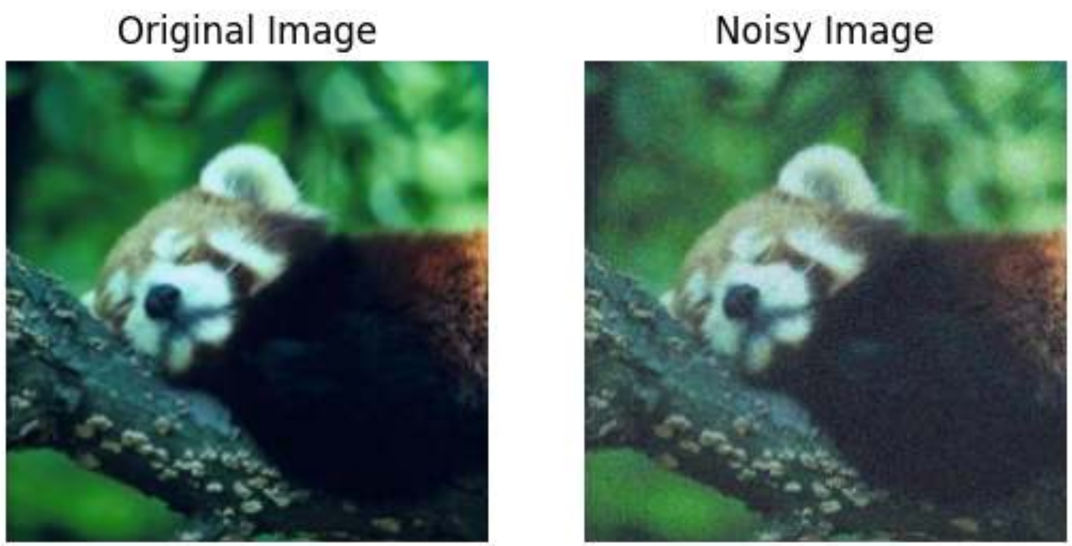

Original Image
Noisy Image
Original Image
Noisy Image


Original Image
Noisy Image
Original Image
Noisy Image


Original Image
Noisy Image
Original Image
Noisy Image


Original Image
Noisy Image
Original Image
Noisy Image





The images look a bit noisier but still very recognizable, and the accuracy is 90% or 100% (depending on the run).

Performing the PGD attack

the difference shown between images is min-max scaled for better visualization. the real difference is very tibny and not visible when directly plotted.

In [16]:
```
for param in model.parameters():
    param.requires_grad = False
```

In [17]:
```
def add_targeted_fgsm_to_image(img, img_grad, epsilon=0.2, sign: bool=True):
    if sign:
        img_fgsm = img - epsilon * img_grad.sign()
    else:
        img_fgsm = img + epsilon * img_grad
    # img_fgsm = torch.clamp(img_fgsm, 0, 1)
    return img_fgsm
```

In [18]:
```
def report_losses_targeted_pgd_image(model: nn.Module, img: torch.tensor, ground_truth: torch.tensor, new_t
    model.eval()
    img = img.to(device)
    img_copy = img.clone().detach().to(device)
    new_target = new_target.to(device)
    ground_truth = ground_truth.to(device)
    img_copy.requires_grad = True
    loss_1 = None
    pred_class = torch.LongTensor([-1])

    for _ in range(num_steps):
        model.zero_grad()
        y_hat = model(img_copy)

        if y_hat.argmax(1) == new_target:
            print(f"done in {_} steps")
            break

        if pred_class.item() == -1:
            pred_class = nn.Softmax(1)(y_hat).argmax(1)
        loss = nn.CrossEntropyLoss()(y_hat, new_target)
        if not loss_1:
```

```
            loss_1 = nn.CrossEntropyLoss()(y_hat, ground_truth)

        loss.backward()
        img_grad = img_copy.grad.data

        img_fgsm = add_targeted_fgsm_to_image(img_copy, img_grad, epsilon)
        img_copy = img_fgsm.clone().detach().to(device)
        img_copy.requires_grad = True

    y_hat_pgd = model(img_copy)
    predicted_class_pgd = nn.Softmax(1)(y_hat_pgd).argmax(1)
    model.zero_grad()

    return {
        "ground_truth": ground_truth.item(),
        "new_target": new_target.item(),
        "predicted_class": pred_class.item(),
        "pgd_predicted_class": predicted_class_pgd.item(),
        "loss_x,yog": loss_1,
        "img": img.squeeze().detach().cpu().numpy(),
        "pgd_img": img_copy.squeeze().detach().cpu().numpy(),
        "diff": (img_copy - img).squeeze().detach().cpu().numpy()
    }
```

In [19]:
```
def plot_pgd_results(pgd_data):
    def zero_one_rescale(img):
        img = (img - img.min()) / (img.max() - img.min())
        return img

    fig, axs = plt.subplots(1, 3, figsize=(20, 10))
    axs[0].imshow(min_max_scale_img(einops.rearrange(torch.tensor(pgd_data["img"]), 'c h w -> h w c')))
    axs[0].set_title(f"Original Image, GT: {CLASSES[pgd_data['ground_truth']]}, Pred: {CLASSES[pgd_data['pr
    pgd_img = einops.rearrange(torch.tensor(pgd_data["pgd_img"]), 'c h w -> h w c')
    axs[1].imshow(min_max_scale_img(pgd_img))
    axs[1].set_title(f"PGD Image, Target: {CLASSES[pgd_data['new_target']]}, Pred: {CLASSES[pgd_data['pgd_p
    # axs[2].imshow(min_max_scale_img(einops.rearrange(torch.tensor(pgd_data["diff"]), 'c h w -> h w c')))
    axs[2].imshow(zero_one_rescale(einops.rearrange(torch.tensor(pgd_data["diff"]), 'c h w -> h w c')))
    axs[2].set_title("Difference")
```

```
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def perform_attack(model, dataset: Dataset, num_steps: int=3, epsilon: float=0.1, second_most_probable:bool
    model.eval()
    model.to(device)

    dataloader = DataLoader(dataset, batch_size=1, shuffle=False)
    results = []

    for img, label in dataloader:
        img = img.to(device)
        label = label.to(device)
        y_hat = model(img)
        y_hat = nn.Softmax(dim=1)(y_hat)
        y_hat = y_hat.squeeze()
        if second_most_probable:
            y_hat[label] = 0
            new_target = y_hat.argmax()
        else:
            new_target = y_hat.argmin()
        results.append(report_losses_targeted_pgd_image(model, img, label, new_target.unsqueeze(0), num_ste

    return results
```

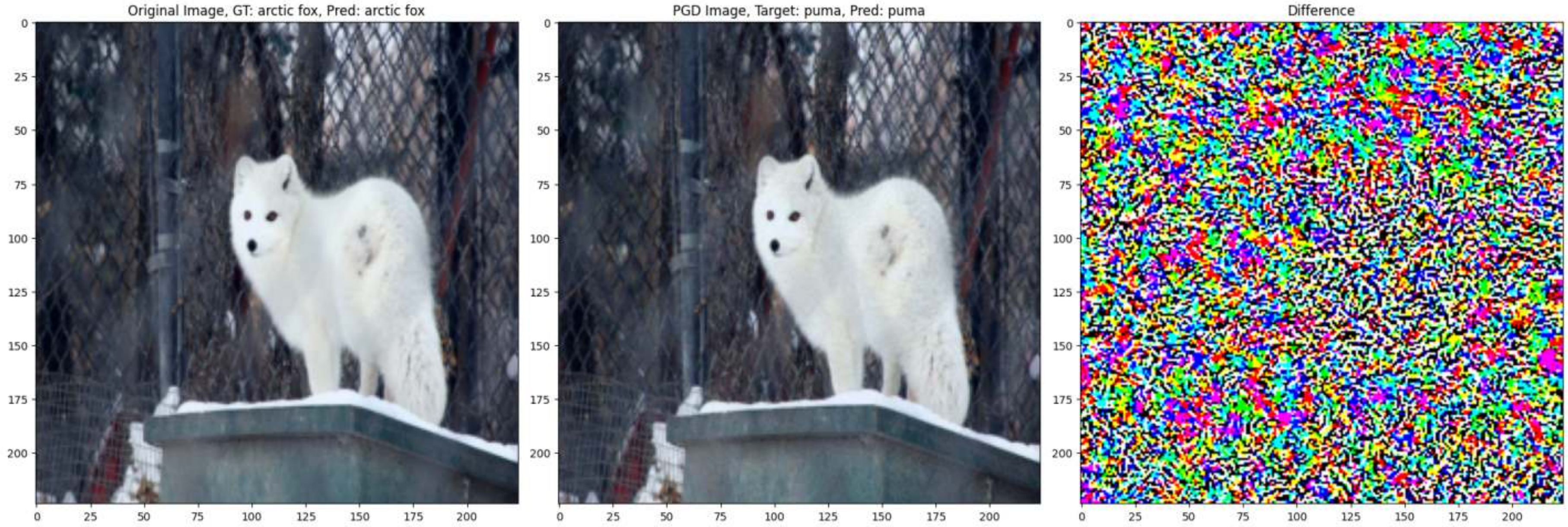
targeting the second most probable class

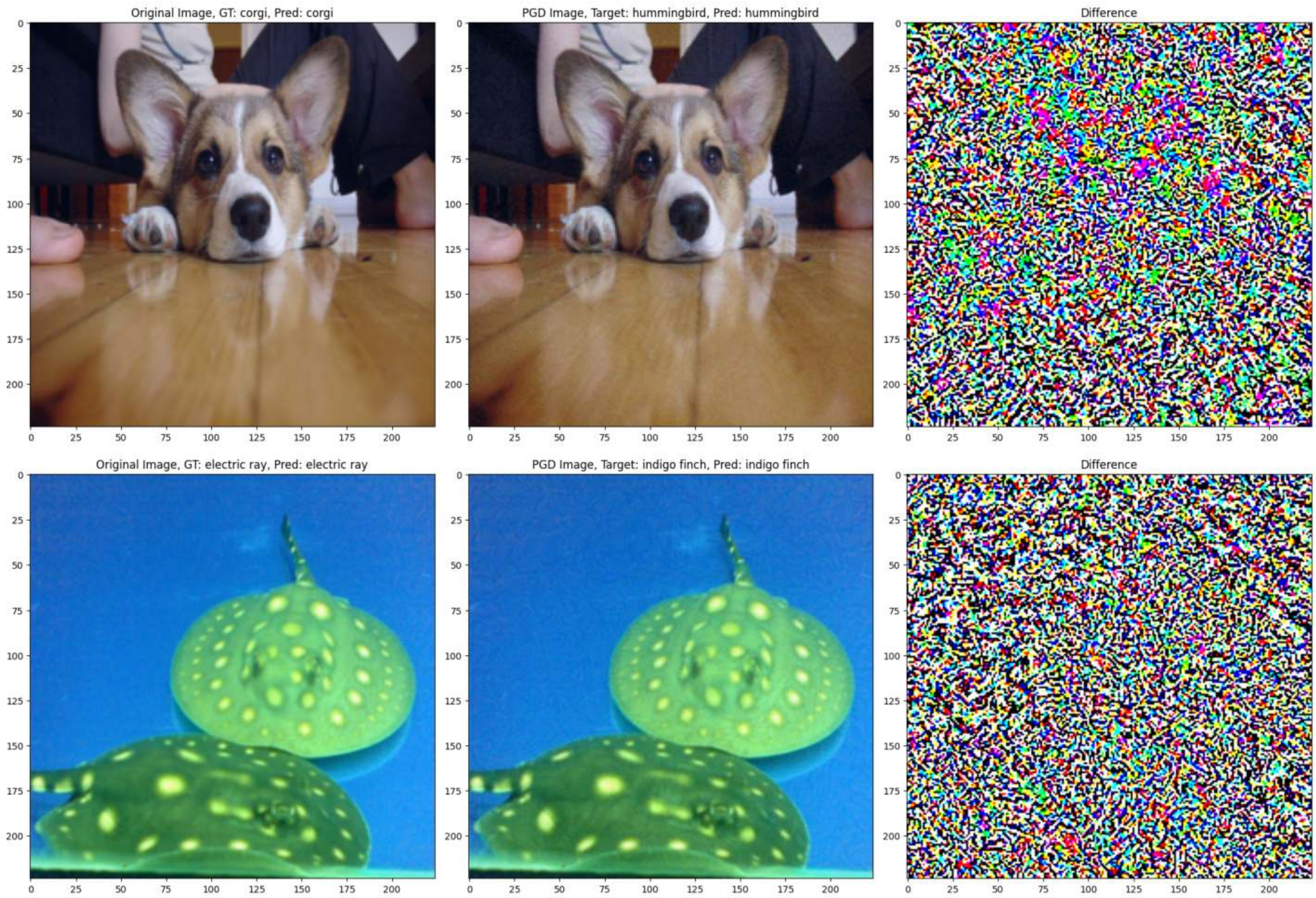
In [21]:
```
pgd_attack_results = perform_attack(model, cute_dataset, num_steps=200, epsilon=0.05)
len(pgd_attack_results)
```

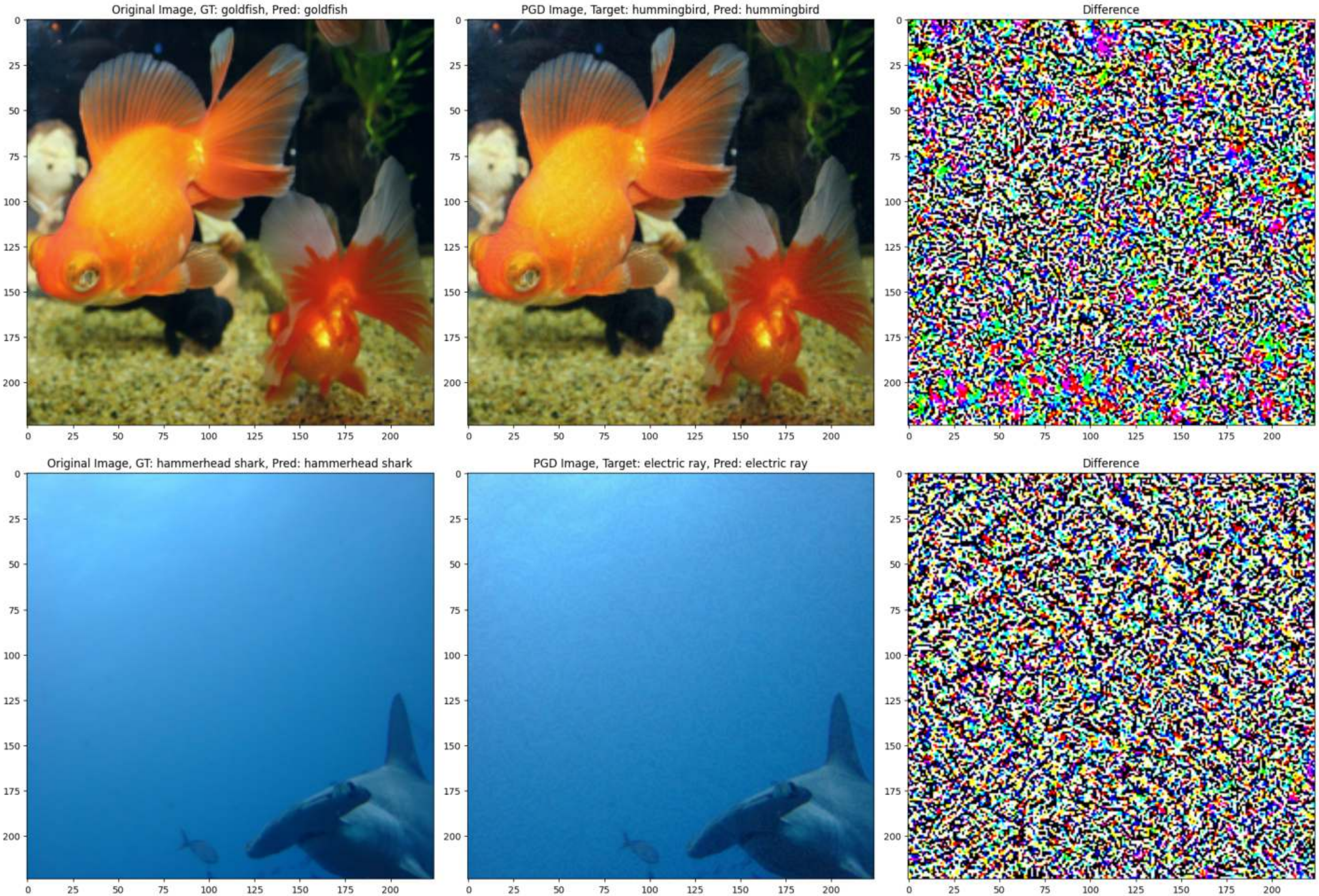
```
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
done in 1 steps
```

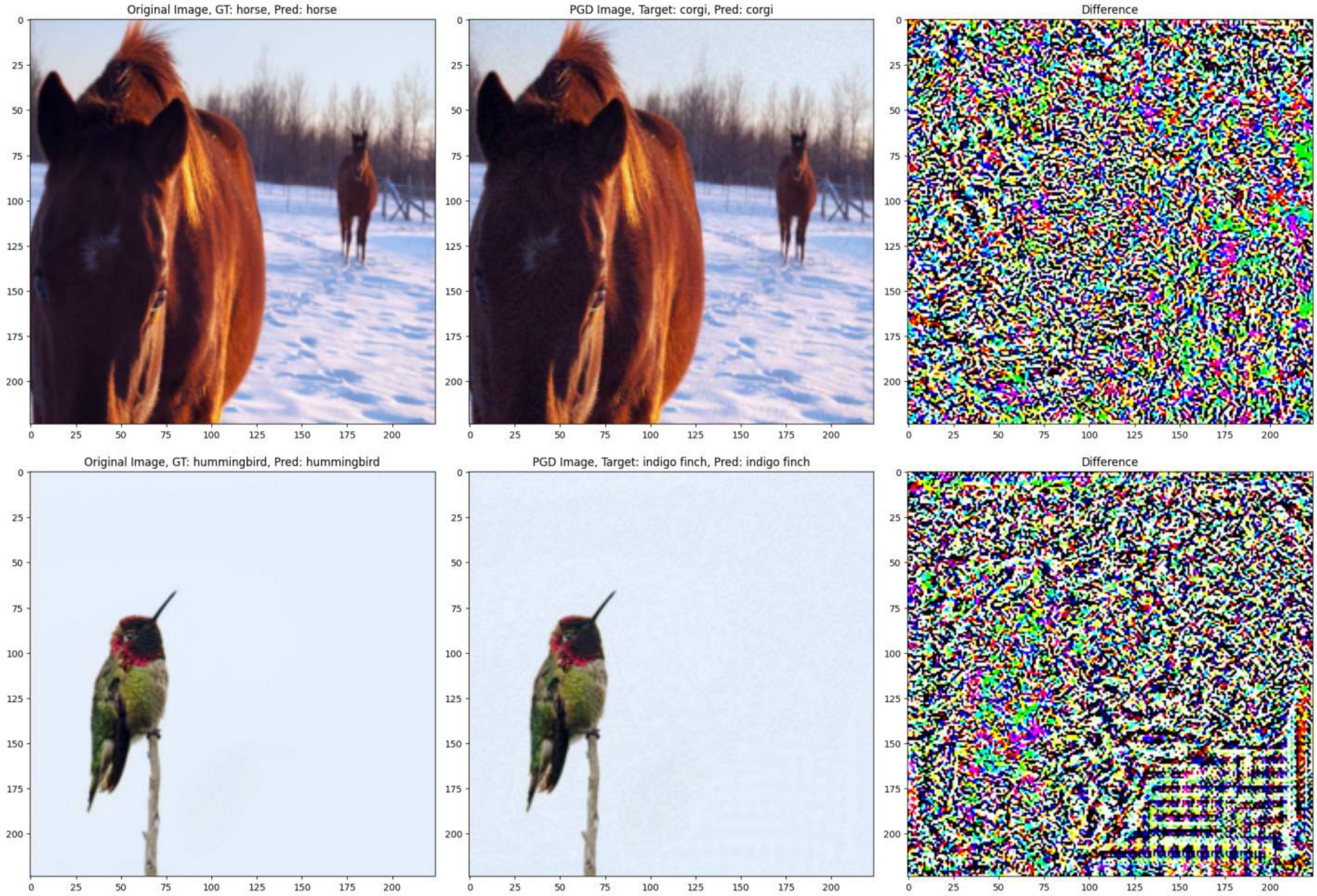
Out[21]: 10

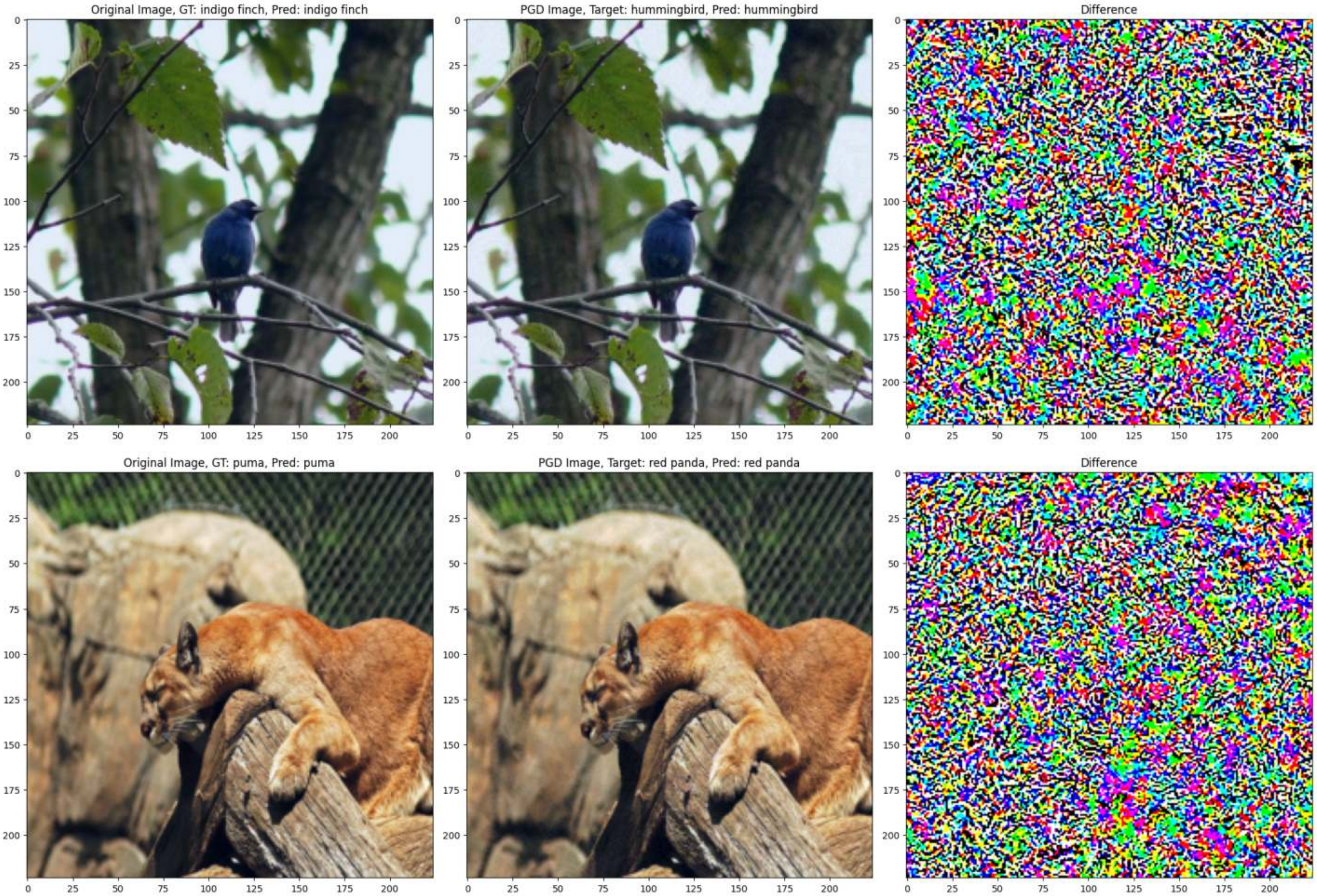
In [ ]:
```python
for attack_data in pgd_attack_results:
    plot_pgd_results(attack_data)
```

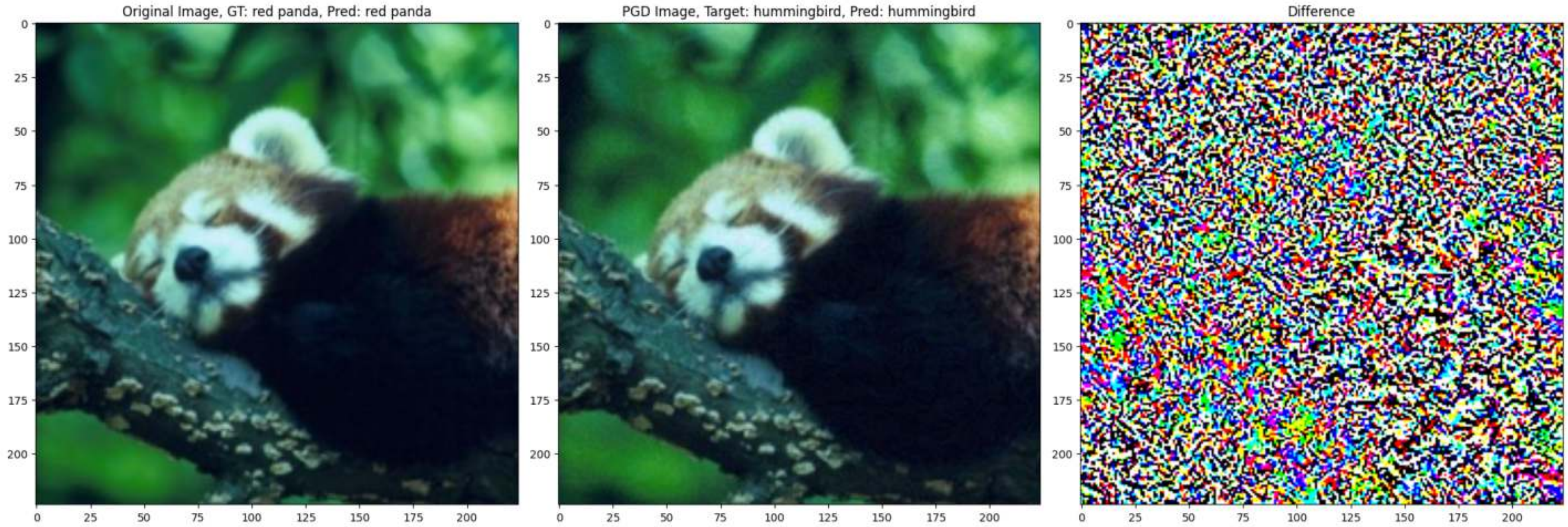



Original Image, GT: corgi, Pred: corgi
PGD Image, Target: hummingbird, Pred: hummingbird
Difference
Original Image, GT: electric ray, Pred: electric ray
PGD Image, Target: indigo finch, Pred: indigo finch
Difference


Original Image, GT: goldfish, Pred: goldfish
PGD Image, Target: hummingbird, Pred: hummingbird
Difference
Original Image, GT: hammerhead shark, Pred: hammerhead shark
PGD Image, Target: electric ray, Pred: electric ray
Difference


Original Image, GT: horse, Pred: horse
PGD Image, Target: corgi, Pred: corgi
Difference
Original Image, GT: hummingbird, Pred: hummingbird
PGD Image, Target: indigo finch, Pred: indigo finch
Difference


Original Image, GT: indigo finch, Pred: indigo finch
PGD Image, Target: hummingbird, Pred: hummingbird
Difference
Original Image, GT: puma, Pred: puma
PGD Image, Target: red panda, Pred: red panda
Difference
0
25
50
75
100
125
150
175
200



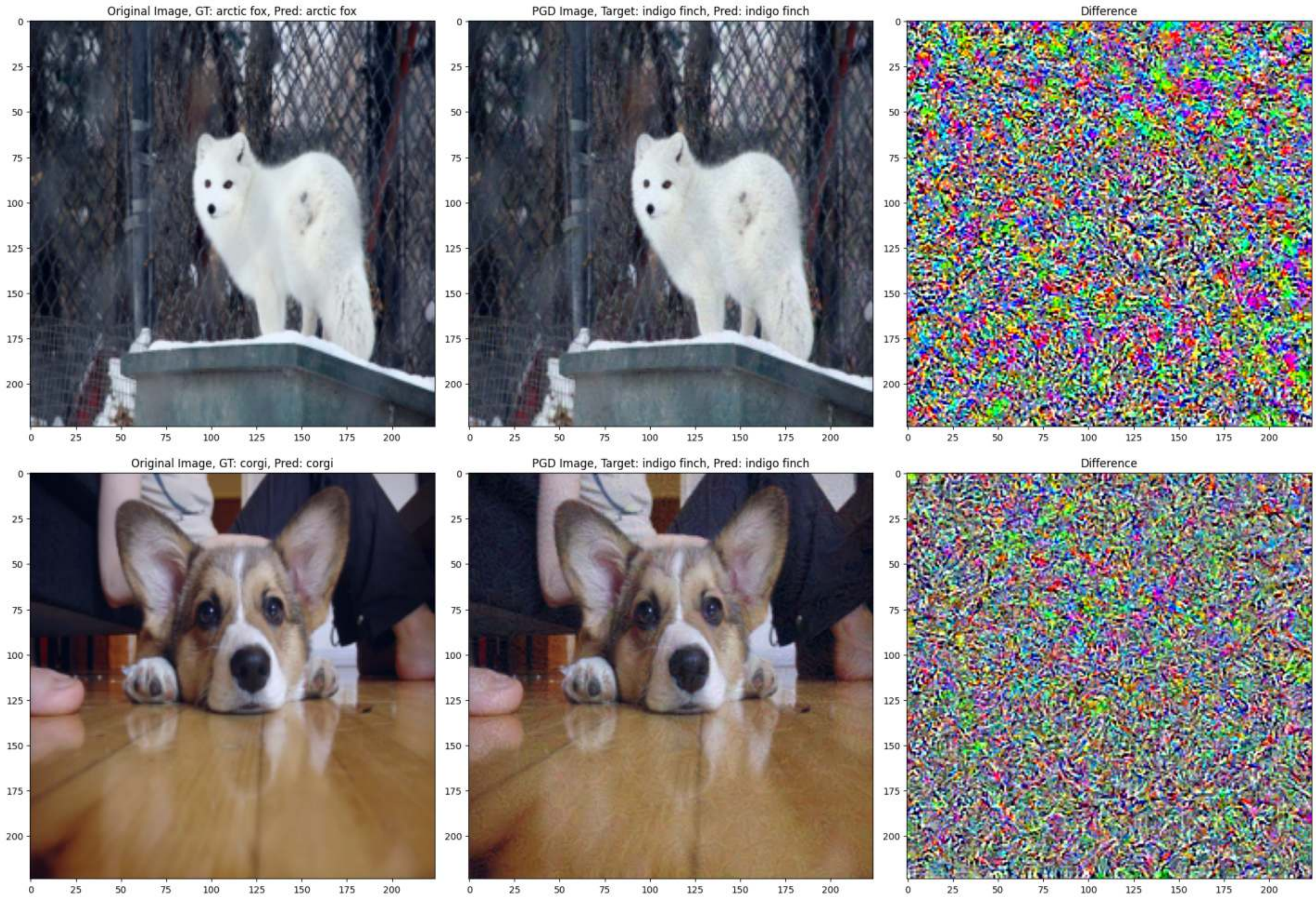
targeting the least likely class

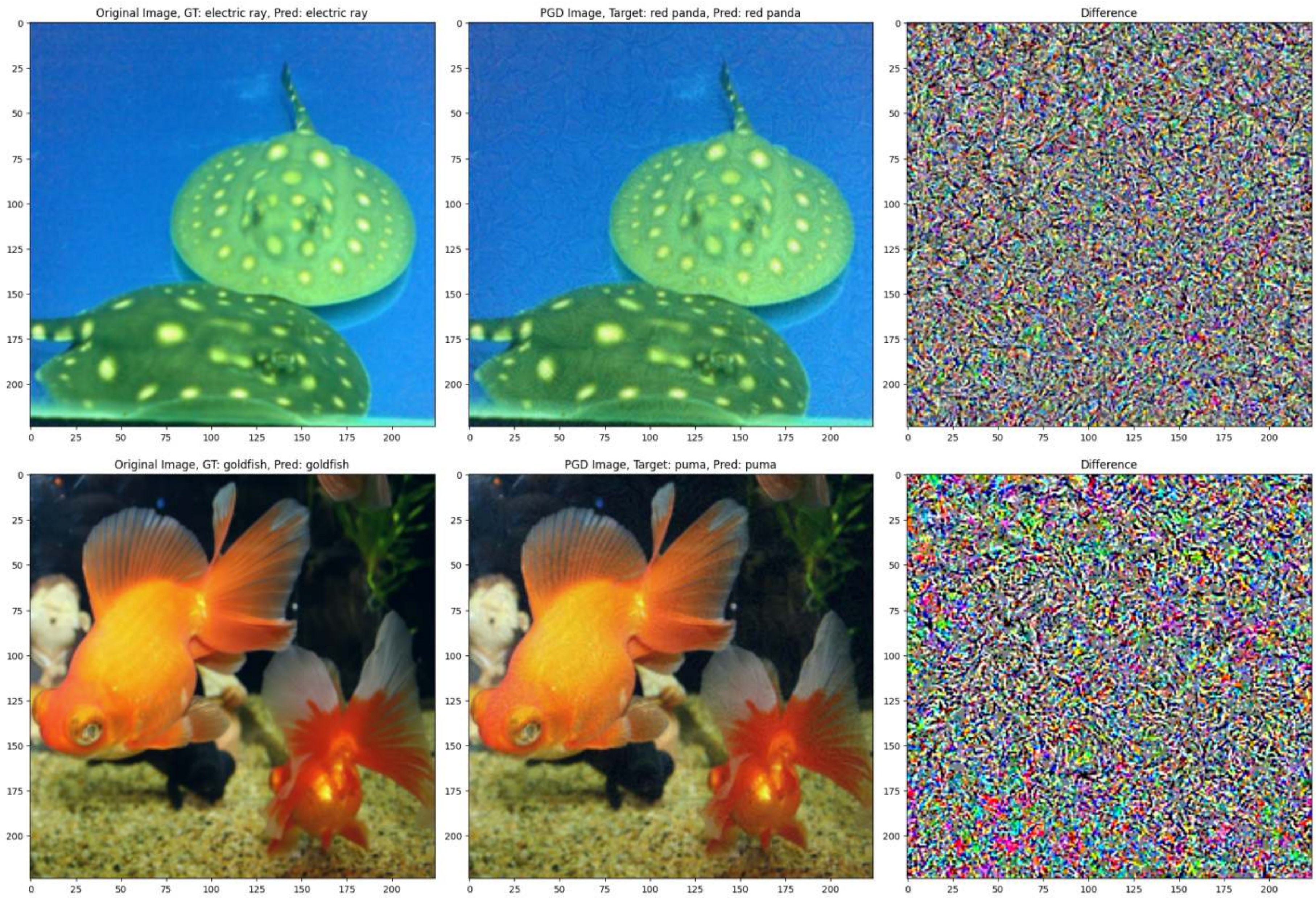
In [23]:
```
pgd_attack_results_least_likely = perform_attack(model, cute_dataset, num_steps=100, epsilon=0.05, second_m
len(pgd_attack_results_least_likely)
```

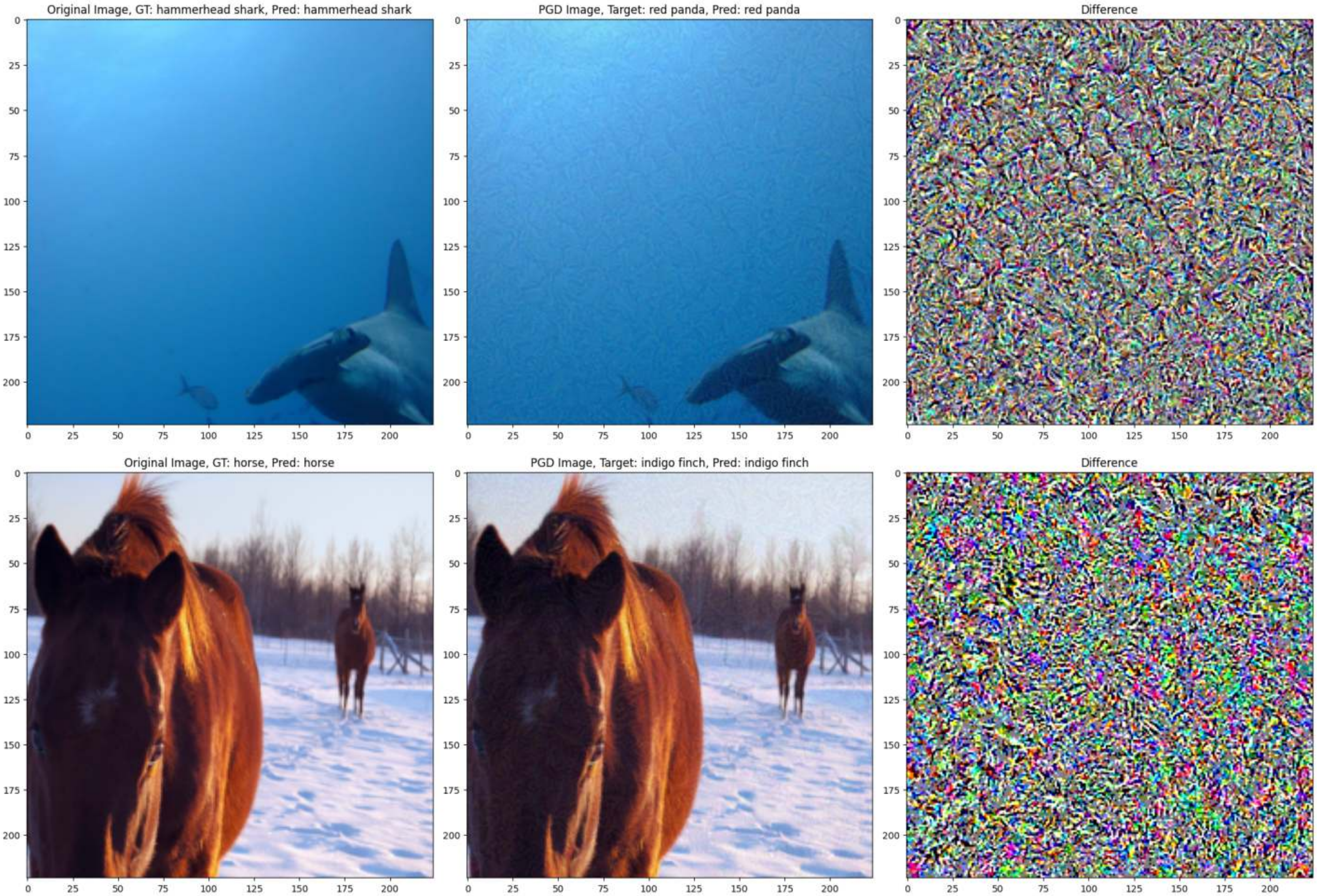
```
done in 2 steps
done in 3 steps
done in 3 steps
done in 2 steps
done in 3 steps
done in 2 steps
done in 2 steps
done in 2 steps
done in 2 steps
done in 2 steps
```

Out[23]: 10

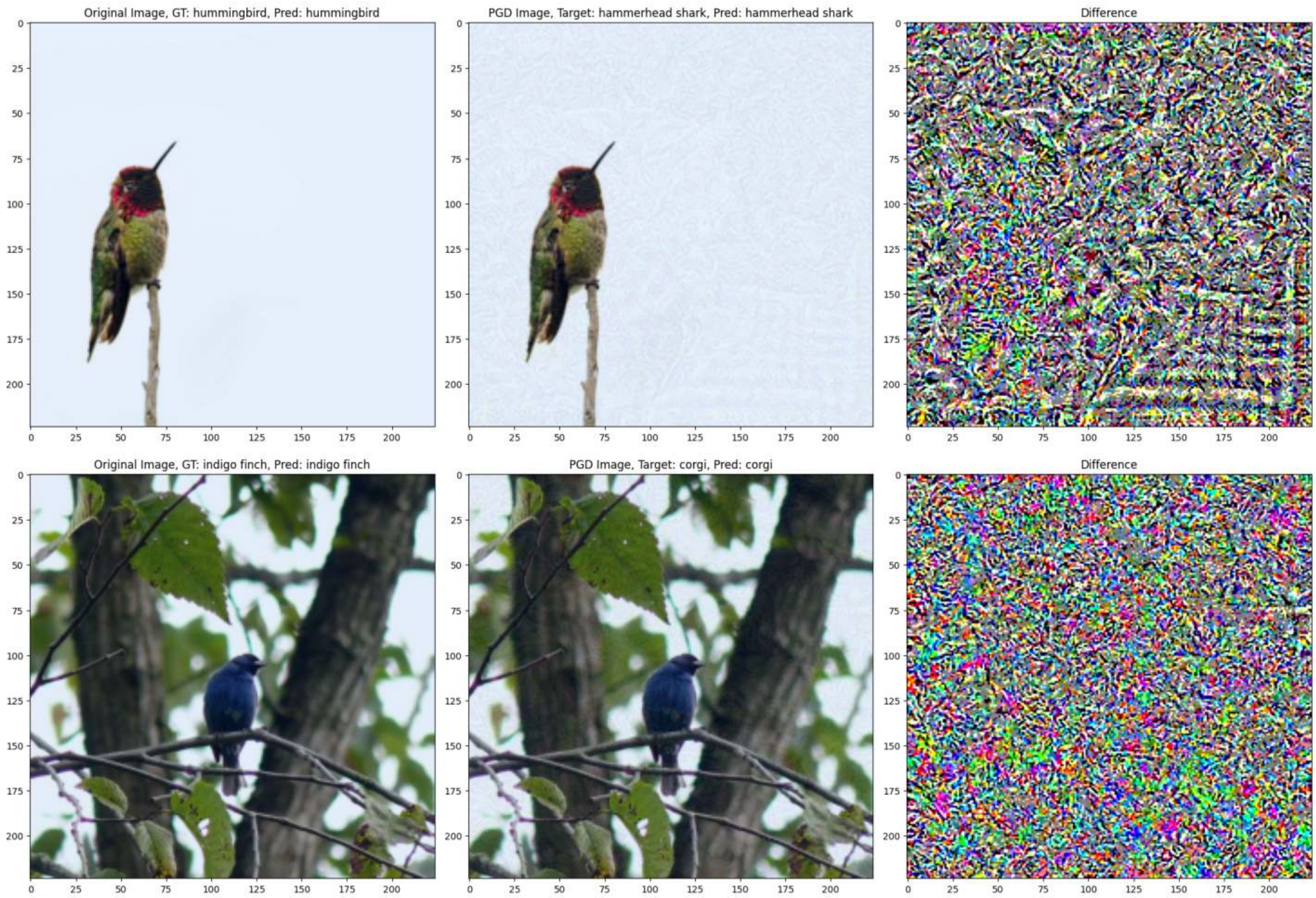
In [ ]:
```
for attack_data in pgd_attack_results_least_likely:
    plot_pgd_results(attack_data)
```


Original Image, GT: arctic fox, Pred: arctic fox
PGD Image, Target: indigo finch, Pred: indigo finch
Difference
Original Image, GT: corgi, Pred: corgi
PGD Image, Target: indigo finch, Pred: indigo finch
Difference


Original Image, GT: electric ray, Pred: electric ray
PGD Image, Target: red panda, Pred: red panda
Difference
Original Image, GT: goldfish, Pred: goldfish
PGD Image, Target: puma, Pred: puma
Difference



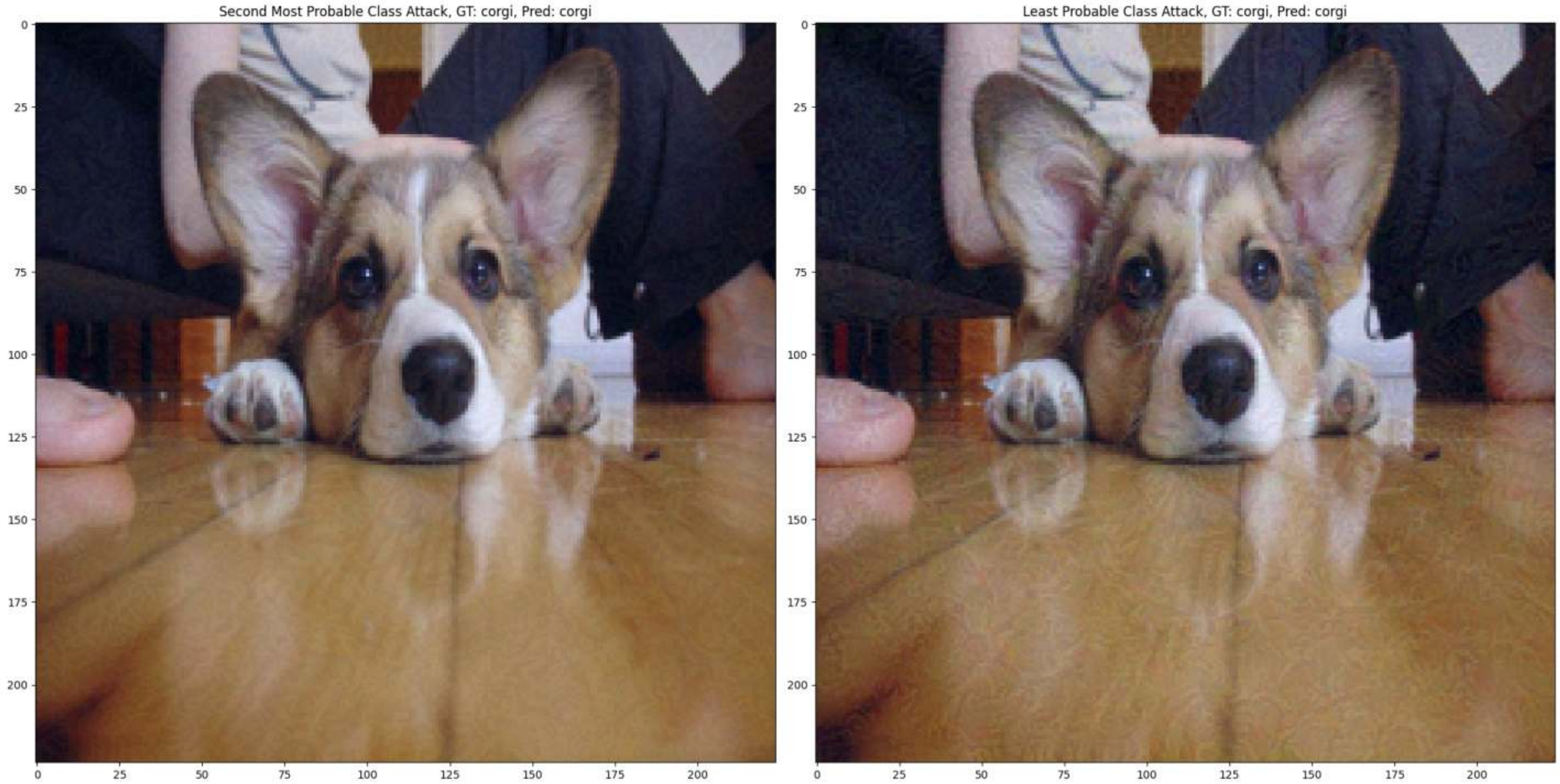
Original Image, GT: hammerhead shark, Pred: hammerhead shark
PGD Image, Target: red panda, Pred: red panda
Difference
Original Image, GT: horse, Pred: horse
PGD Image, Target: indigo finch, Pred: indigo finch
Difference

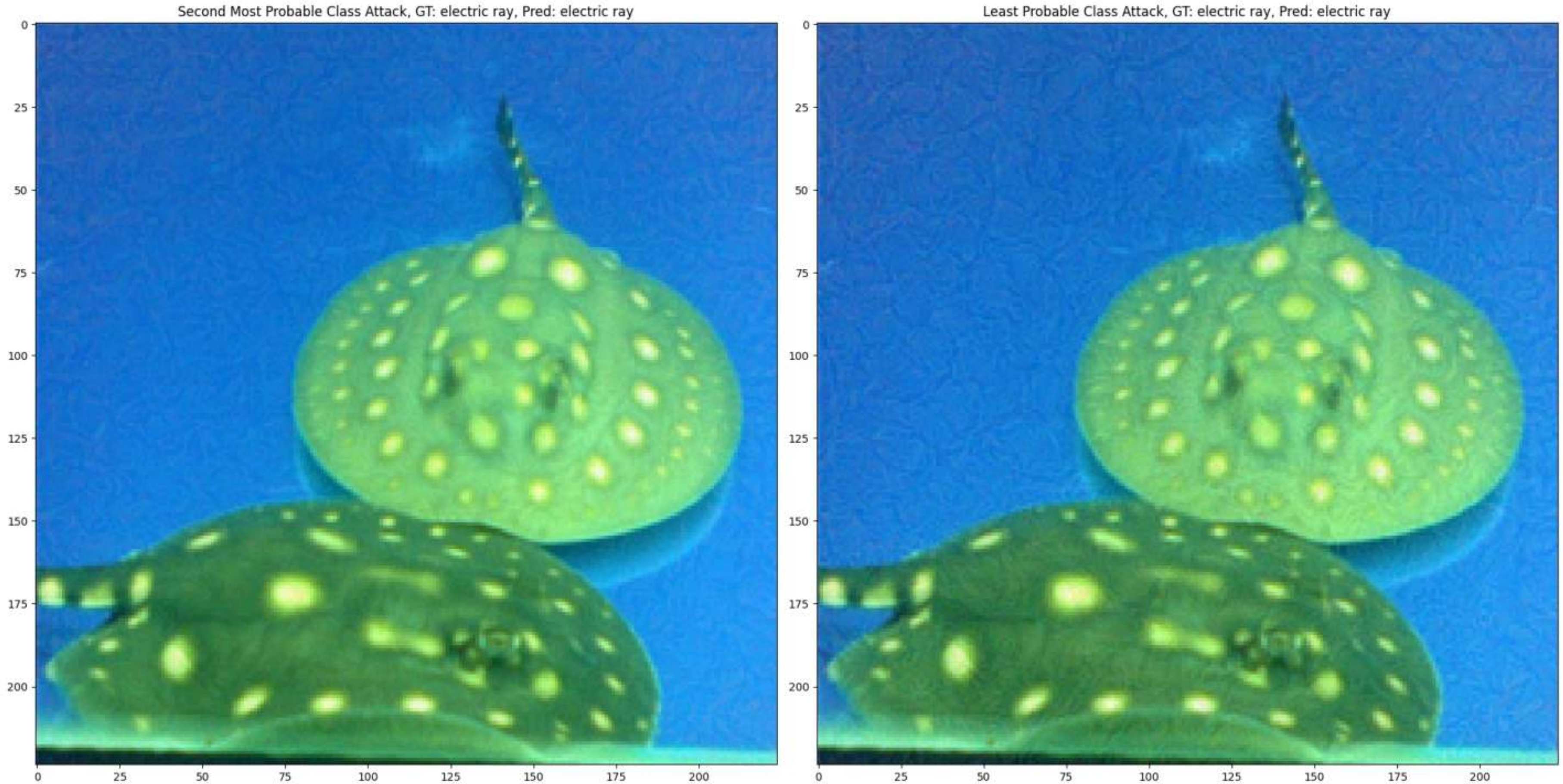

Original Image, GT: hummingbird, Pred: hummingbird
PGD Image, Target: hammerhead shark, Pred: hammerhead shark
Difference
Original Image, GT: indigo finch, Pred: indigo finch
PGD Image, Target: corgi, Pred: corgi
Difference
0
25
50
75
100
125
150
175
200

comparing the final images

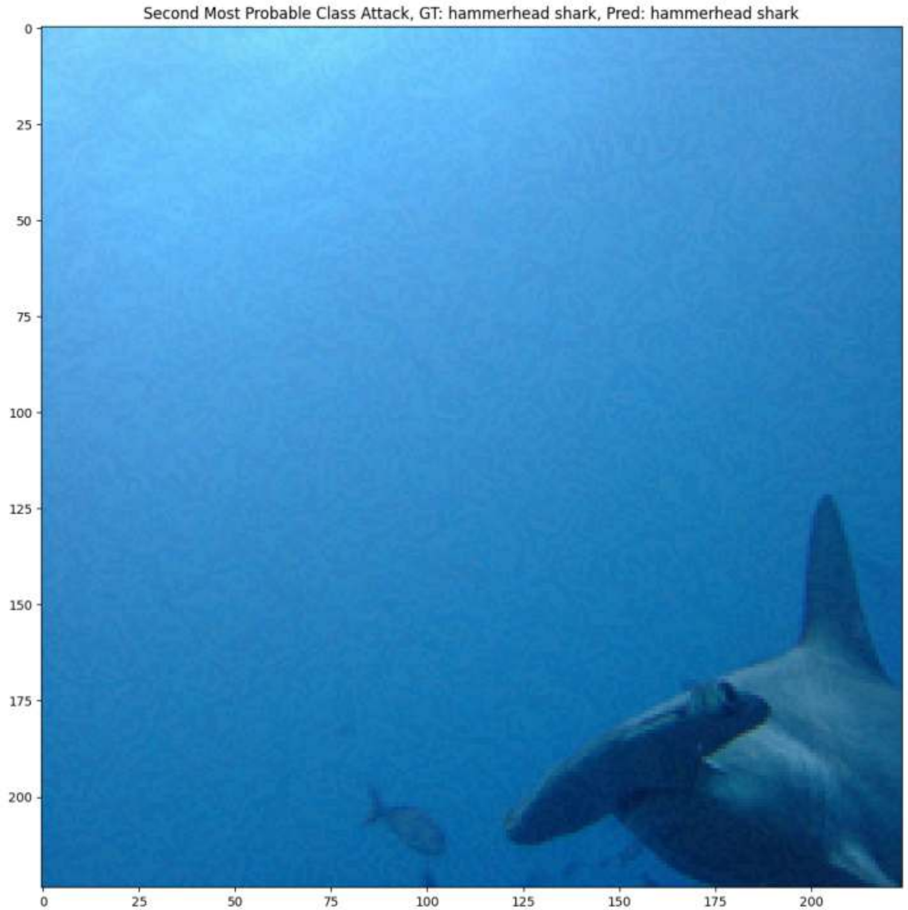
In [28]:
```python
def plot_differences(second_most_prob_data, least_prob_data):
    fig, axs = plt.subplots(1, 2, figsize=(20, 10))
    axs[0].imshow(min_max_scale_img(einops.rearrange(torch.tensor(second_most_prob_data["pgd_img"]), 'c h w
    axs[0].set_title(f"Second Most Probable Class Attack, GT: {CLASSES[second_most_prob_data['ground_truth'
    axs[1].imshow(min_max_scale_img(einops.rearrange(torch.tensor(least_prob_data["pgd_img"]), 'c h w -> h
    axs[1].set_title(f"Least Probable Class Attack, GT: {CLASSES[least_prob_data['ground_truth']]}, Pred: {
    plt.tight_layout()
    plt.show()
```

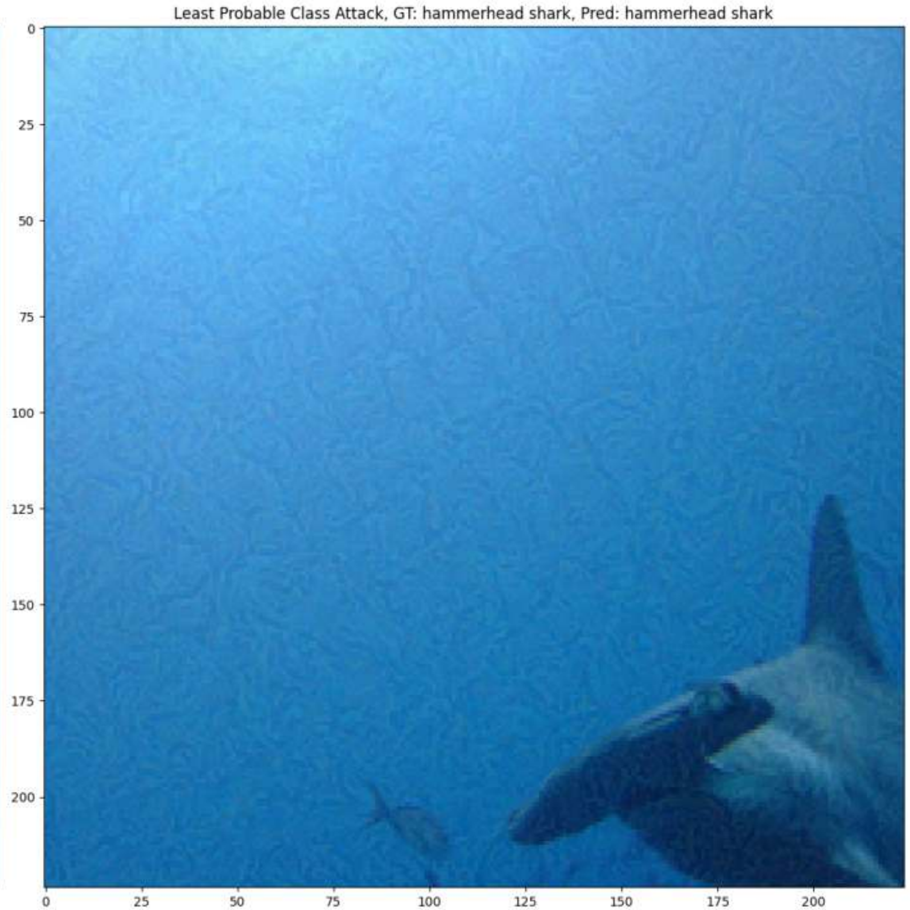
In [29]:
```python
for i in range(len(pgd_attack_results)):
    plot_differences(pgd_attack_results[i], pgd_attack_results_least_likely[i])
```

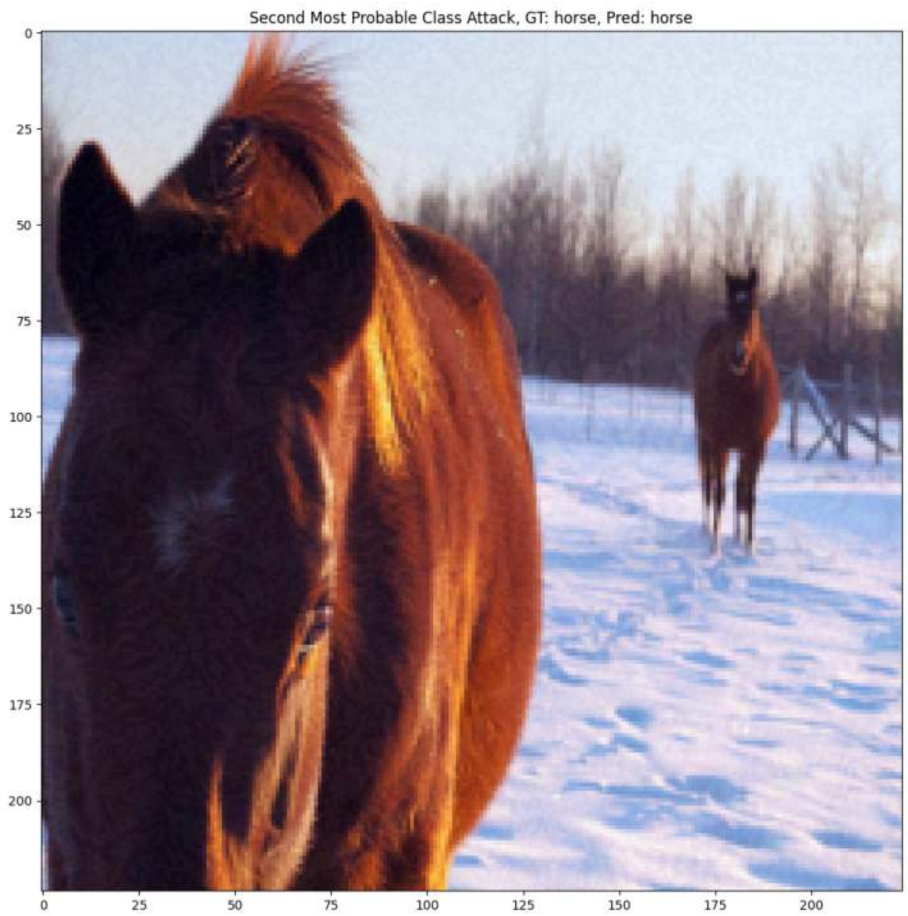

Second Most Probable Class Attack, GT: corgi, Pred: corgi
Least Probable Class Attack, GT: corgi, Pred: corgi
0
25
50
75
100
125
150
175
200

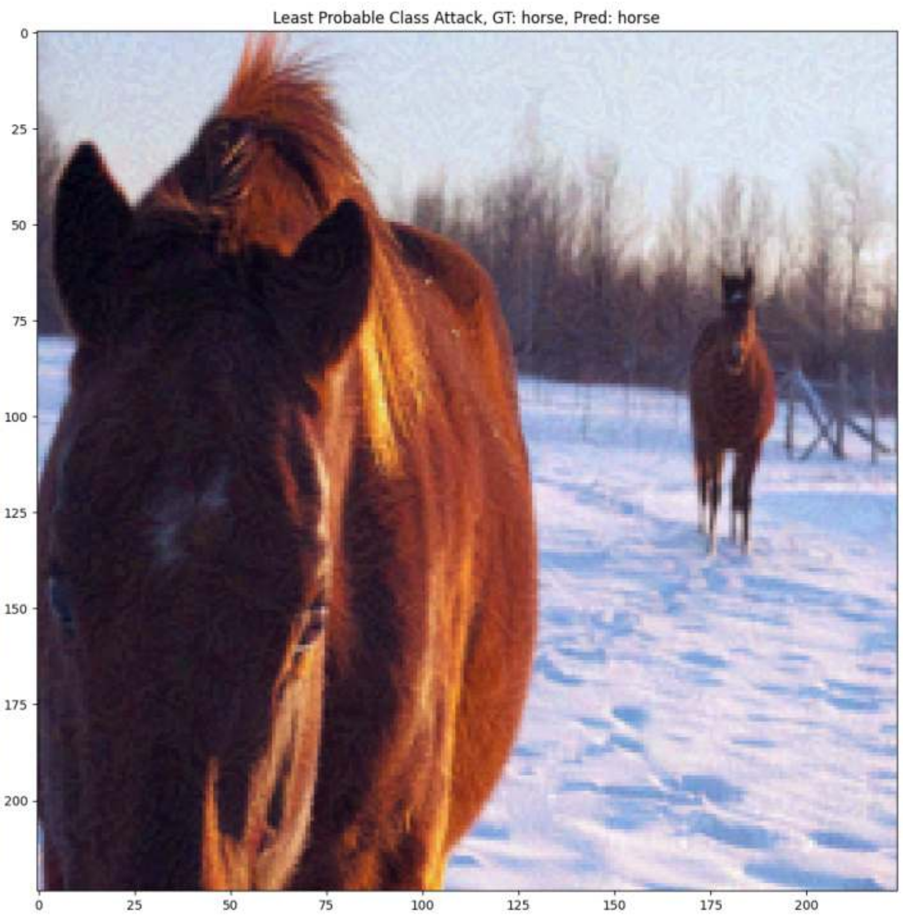

Second Most Probable Class Attack, GT: electric ray, Pred: electric ray
Least Probable Class Attack, GT: electric ray, Pred: electric ray
0
25
50
75
100
125
150
175
200

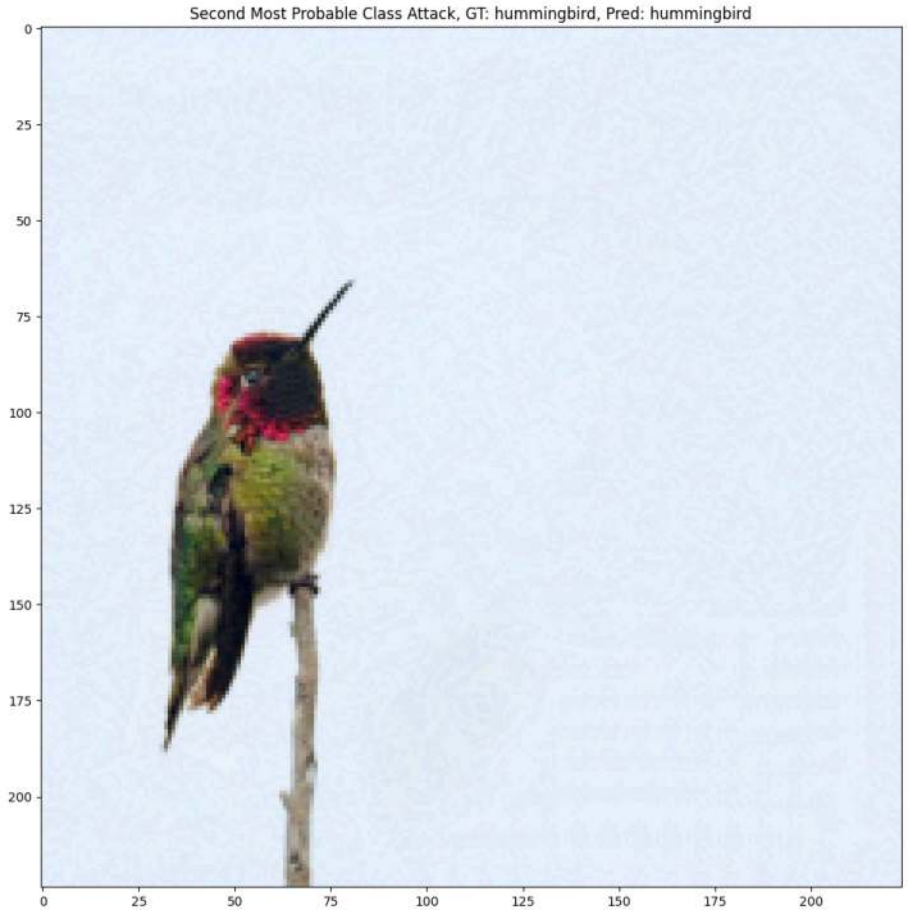

Second Most Probable Class Attack, GT: goldfish, Pred: goldfish
Least Probable Class Attack, GT: goldfish, Pred: goldfish

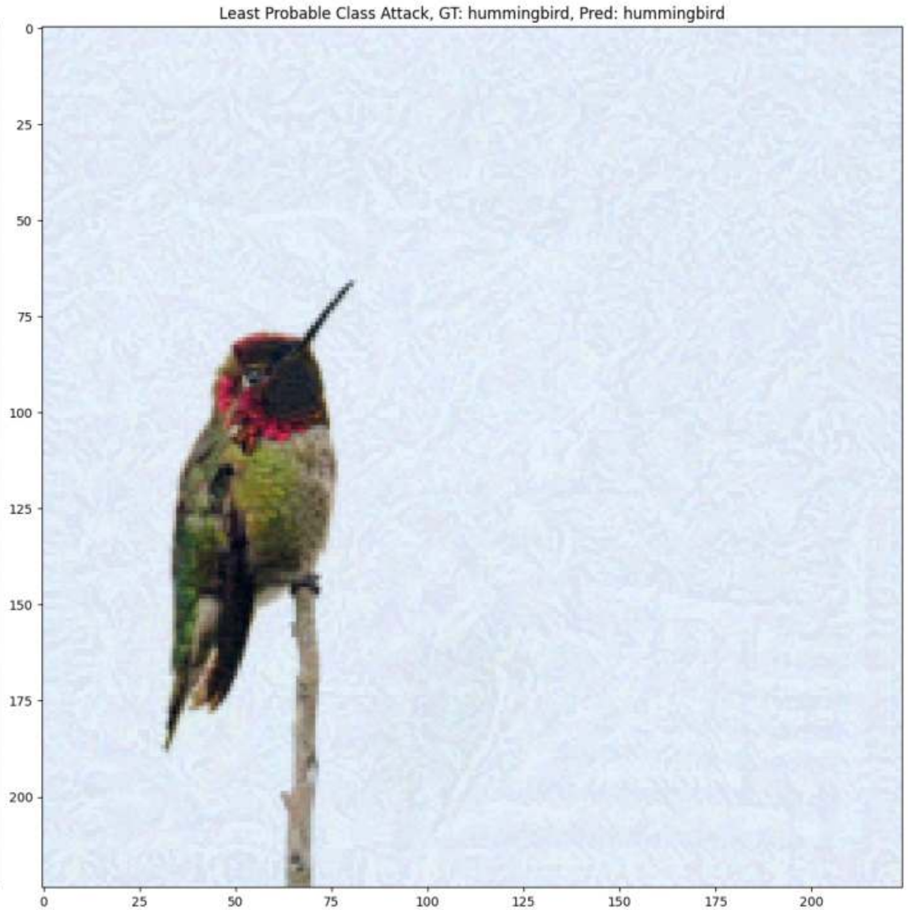

Second Most Probable Class Attack, GT: hammerhead shark, Pred: hammerhead shark

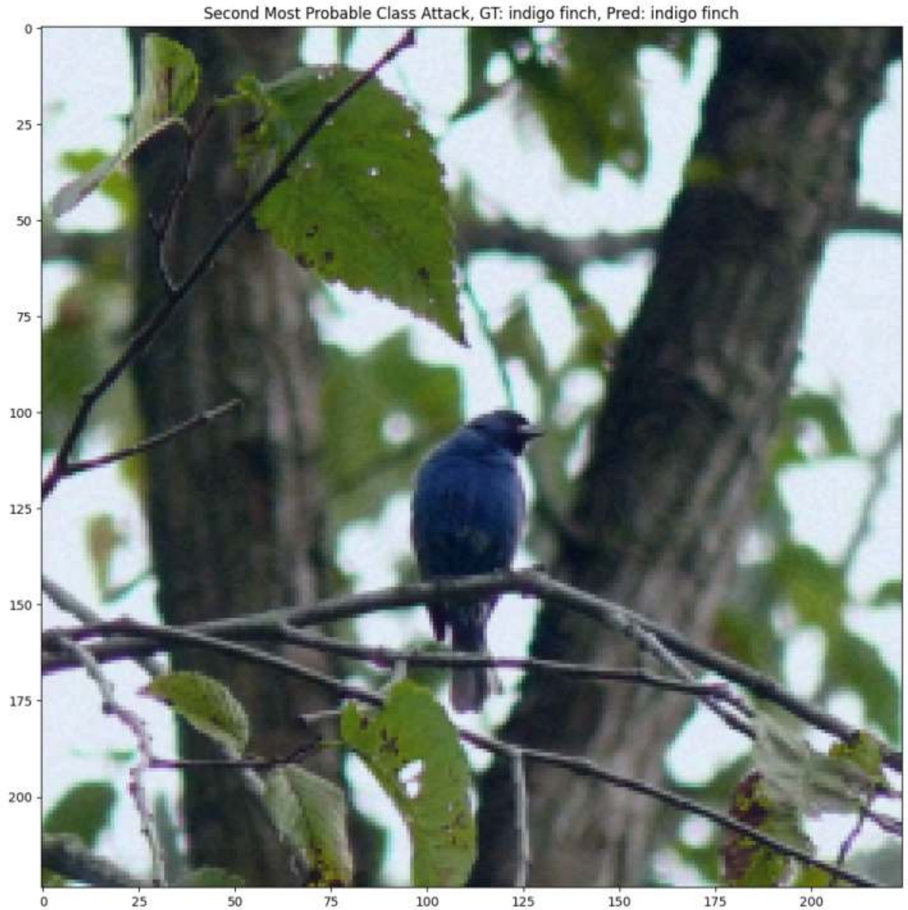

Least Probable Class Attack, GT: hammerhead shark, Pred: hammerhead shark

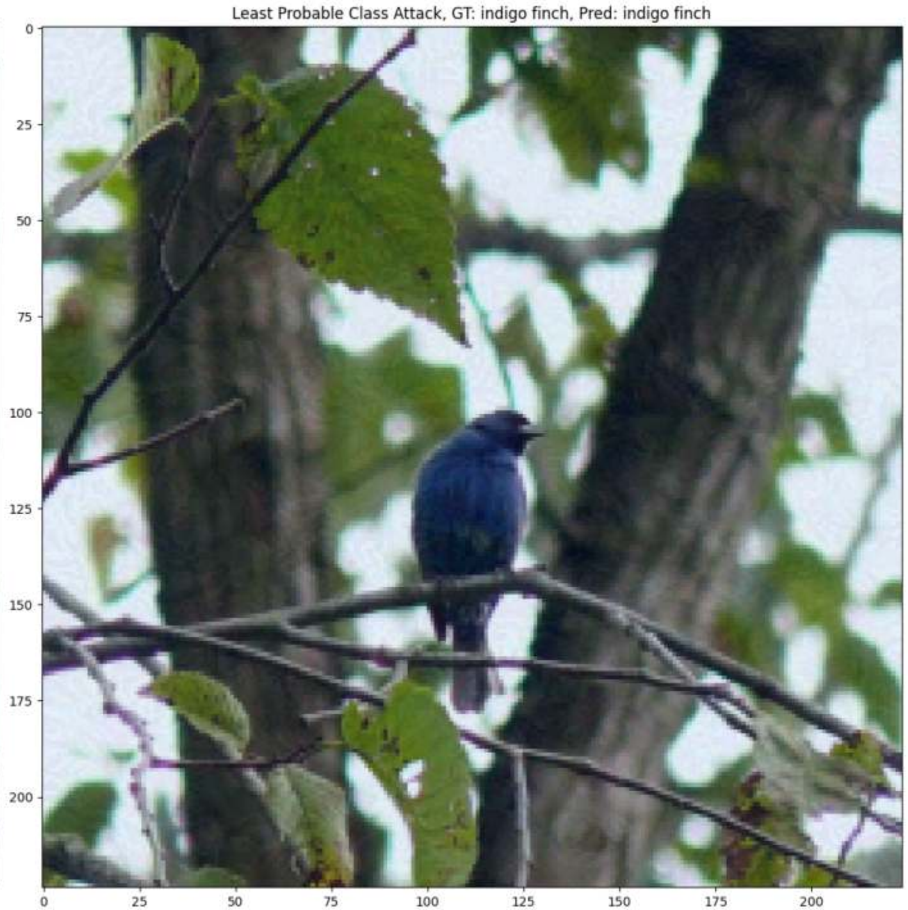

Second Most Probable Class Attack, GT: horse, Pred: horse
0
25
50
75
100
125
150
175
200

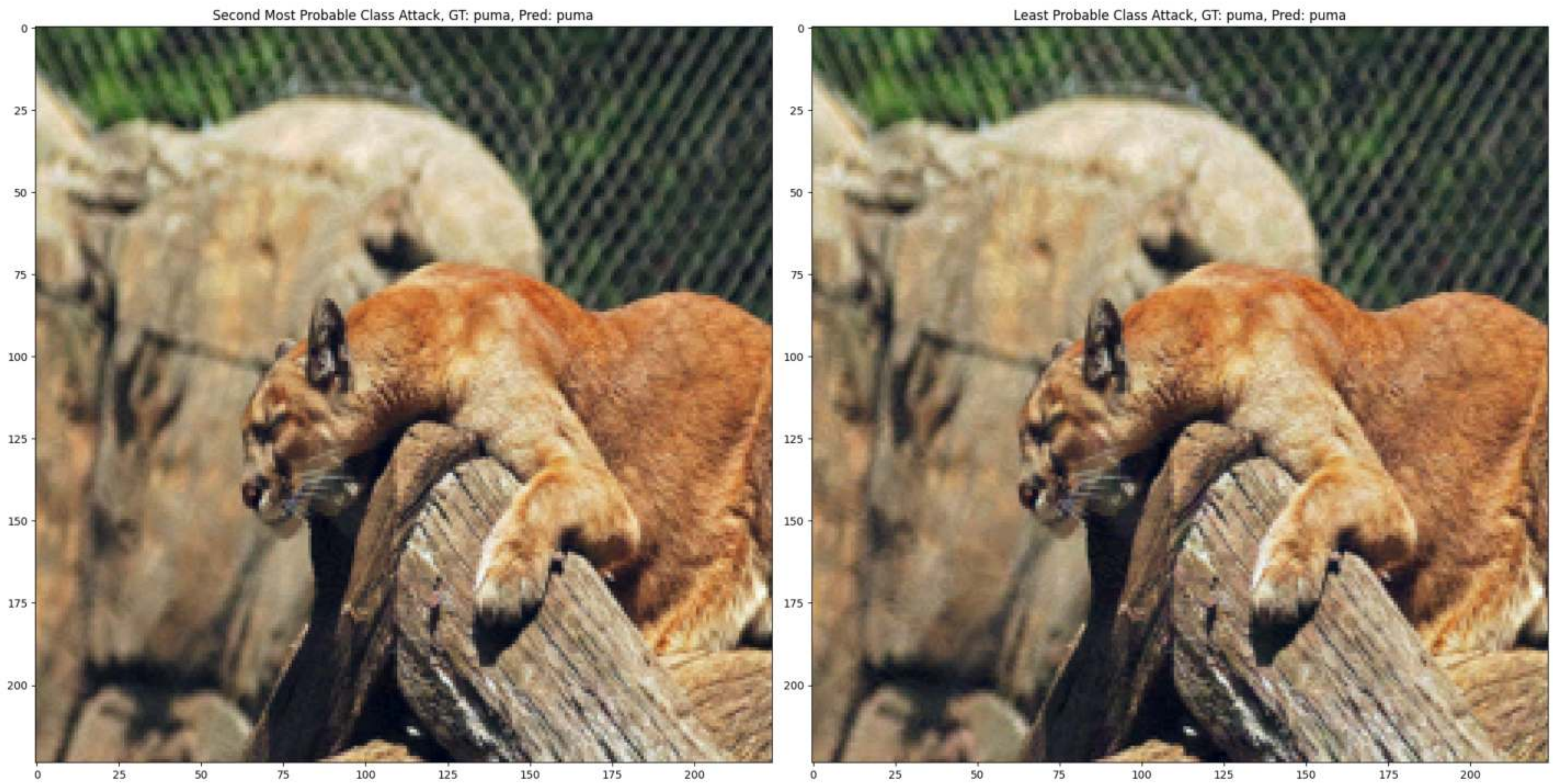

Least Probable Class Attack, GT: horse, Pred: horse
0
25
50
75
100
125
150
175
200


Second Most Probable Class Attack, GT: hummingbird, Pred: hummingbird


Least Probable Class Attack, GT: hummingbird, Pred: hummingbird


Second Most Probable Class Attack, GT: indigo finch, Pred: indigo finch


Least Probable Class Attack, GT: indigo finch, Pred: indigo finch


Second Most Probable Class Attack, GT: puma, Pred: puma
Least Probable Class Attack, GT: puma, Pred: puma

Though the images look quite similar to the original, the pgd image for the least probable class is visibly noisier than the second most probable class. This can be seen especially in the hummingbird image and the water based animals(for the run for which these observations are based), because they need to do a bigger update of 2 or 3 steps than the single step update for the second most probable class.

In [ ]: