MIT CSAIL

6.819/6.869 Advances in Computer Vision

Spring 2022

Problem Set 4

Jiahui Tang

jiahuita@mit.edu

Problem 1 *:Forward Propagation*

(a) **Convolution Layer**

Consider both x_{in} and x_{out} are 1-D signals with the same dimension N, and Kernel W has size k with stride 1. Forward propagation through a convolution layer could be written as:

$$x_{out} = W \circ x_{in}$$

We can express this as a convolution on the range of W, given k is an odd number,

$$x_{out}[i] = \sum_{j=i-\lfloor k/2 \rfloor}^{i+\lfloor k/2 \rfloor} W[j - i + \lfloor k/2 \rfloor] x_{in}[j]$$

, for $i \in [0, ..., N-1]$

Here the $x_{in}[j]$ will be the padding value (we do not assume any padding value, it commonly be zero, but this formula holds for other values too) for $j < 0$ or $j >= N$, and $W[j]$ is the jth element of the kernal W.

Problem 2 *: Neural Network Inference*

(a) In the input of the last layer, there are 2048 features

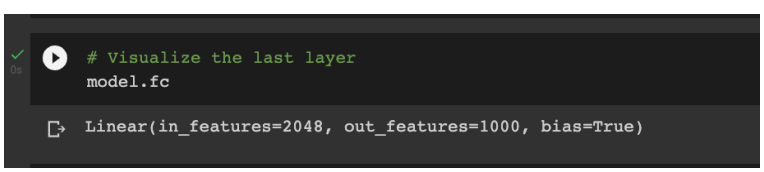

Figure 1: 2a output

(b) Top prediction: racquet

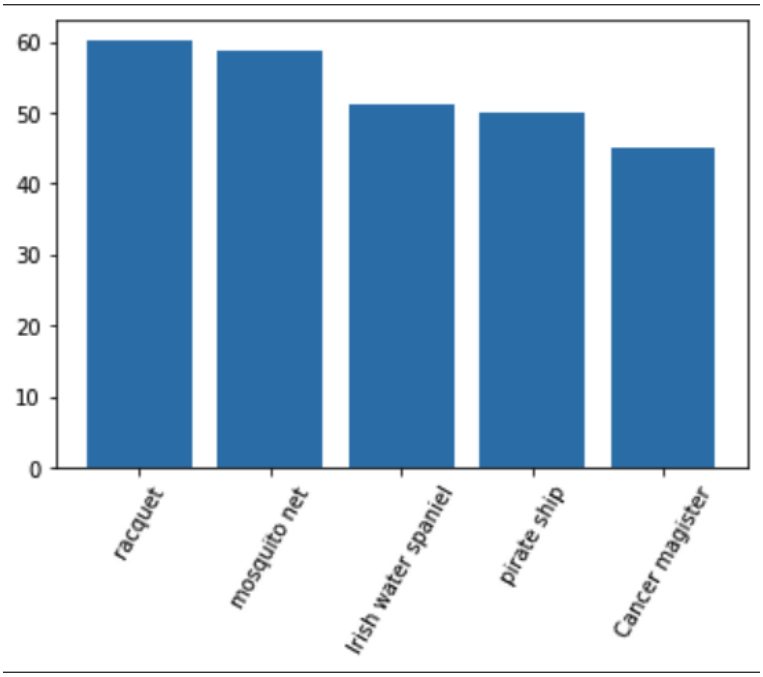

Figure 2: 2b top predictions

(c) Top prediction: Pembroke Welsh Corgi

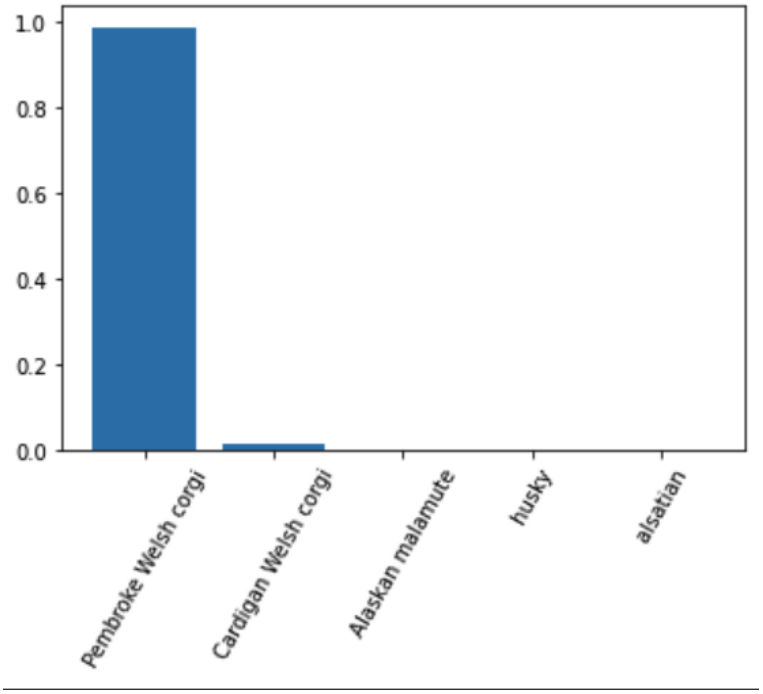

Figure 3: 2c probabilities and top predictions

Probability: 98.6%

```
[174] def output2prob(output):
        ### TODO1
        # Your code here:
        prob = torch.nn.functional.softmax(output)
        ###
        return prob
```

Figure 4: 2c code

```
prob.max()
tensor(0.9864, device='cuda:0', grad_fn=<MaxBackward1>)
```

Figure 5: 2c code

Problem 3 *:Backpropagation*

(a) Equation for backpropagation:

Starting with basic equation for backpropagation

$$\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \frac{\partial x_{out}}{\partial x_{in}}$$

And we have

$$\frac{\partial x_{out}[i]}{\partial x_{in}[j]} = \begin{cases} W_{j-i+\lfloor k/2 \rfloor} & j \in [i - \lfloor k/2 \rfloor, i + \lfloor k/2 \rfloor] \\ 0 & otherwise \end{cases}$$

for all i and $j \in [0, 1, ..., N-1]$

Using matrix form, we can derive relationship as follows,

$$\frac{\partial x_{out}}{\partial x_{in}} = M$$

$=$

$$\begin{bmatrix} W_{\lfloor k/2 \rfloor} & W_{1+\lfloor k/2 \rfloor} & W_{2+\lfloor k/2 \rfloor} & ... & W_{k-1} & 0 & 0 & ... & 0 & 0 \\ W_{\lfloor k/2 \rfloor -1} & W_{\lfloor k/2 \rfloor} & W_{1+\lfloor k/2 \rfloor} & W_{2+\lfloor k/2 \rfloor} & ... & W_{k-1} & 0 & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & W_0 & ... & W_{\lfloor k/2 \rfloor -2} & W_{\lfloor k/2 \rfloor -1} & W_{\lfloor k/2 \rfloor} & W_{\lfloor k/2 \rfloor +1} \\ 0 & 0 & ... & 0 & 0 & W_0 & ... & W_{\lfloor k/2 \rfloor -2} & W_{\lfloor k/2 \rfloor -1} & W_{\lfloor k/2 \rfloor} \end{bmatrix}$$

Using this matrix M, we can express our equation as

$$\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} M$$

We can also express it as

$$x_{out}[i] = \sum_{j=0}^{k-1} W[j] x_{in}[j + i - \lfloor k/2 \rfloor]$$

for all $i \in [0, 1, ..., N-1]$

Thus

$$\frac{\partial x_{out}[i]}{\partial W[j]} = x_{in}[j + i - \lfloor k/2 \rfloor]$$

for all $i \in [0, 1, ..., N-1]$ and $j \in [0, 1, ..., k-1]$

(b) We can start with the chain rule as follows,

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial x_{out}} \frac{\partial x_{out}}{\partial W}$$

given that $\frac{\partial C}{\partial x_{out}}$ is known, we need to find out $\frac{\partial x_{out}}{\partial W}$.

We know that

$$\frac{\partial x_{out}[i]}{\partial W[j]} = x_{in}[j + i - \lfloor k/2 \rfloor]$$

This leads to the update rule:

$$W_j^{t+1} = W_j^t - \eta \frac{\partial C}{\partial x_{out}} \frac{\partial x_{out}}{\partial W} = W_j^t - \eta \sum_{i \in [0,...,N-1]} \frac{\partial C}{\partial x_{out}[i]} x_{in}[j + i - \lfloor k/2 \rfloor]$$

(c)

To handle boundaries, I decide to do zero padding on both sides.

In this way, we still maintain the output shape we want as N, and zero padding will not contribute to the convolution as it won't change the actual calculation. The number of zeros on each side will be $\lfloor \frac{k}{2} \rfloor$, with input size $N + k - 1$ and output size N

(e) For maxpooling, I will handle boundaries by removing boundary values in x_{out}. If we do zero padding, maxpooling could produce value 0 if the values of x_{in} are negative, which impact the result. Thus, we could simply use no padding and do not include boundary in our result, to avoid unwanted behaviour on the boundary.

If we must use padding, we should use a small value such as integer min value.

Problem 4 *: DeepDream and Adversarial Attacks*

Codes for creating adversarial example:

```
def generate_adversarial_example(model_fn, x, class_id, n_iter=200):
    """
    :param model_fn: a callable that takes an input tensor and returns the model logits.
    :param x: input tensor.
    :param class_id: the id of the target class.
    :return: a tensor for the adversarial example
    """
    for i in tqdm(range(n_iter)):
        ### TODO2
        # You should:
        # 1. Run the model with batch_tensor
        # 2. Describe the loss or the objective you want to maximize
        # 3. Compute the gradient of the objective with respect to the image
        # using torch.autograd.grad
        # -
        # Your code here:
        logit = model_fn(x)
        loss = torch.nn.CrossEntropyLoss()
        target = torch.tensor([class_id]).cuda()
        gradient = torch.autograd.grad(loss(logit, target), x)[0]
        ###
        x = step.step(x, -1*gradient)
    return x
```

(a)

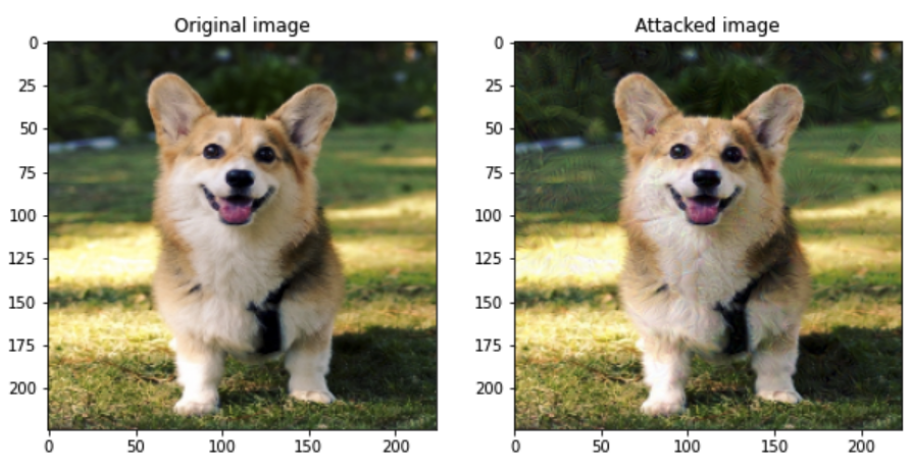

Figure 6: Original Attacked Image of Attack 1 - Tarantula (id 76)

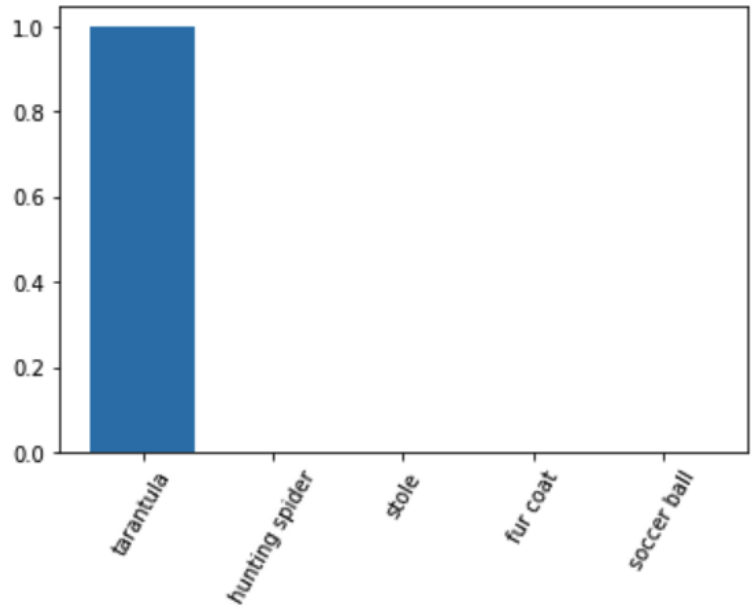

Figure 7: Updated Probability of Attack 1 - Tarantula (id 76)

(b)

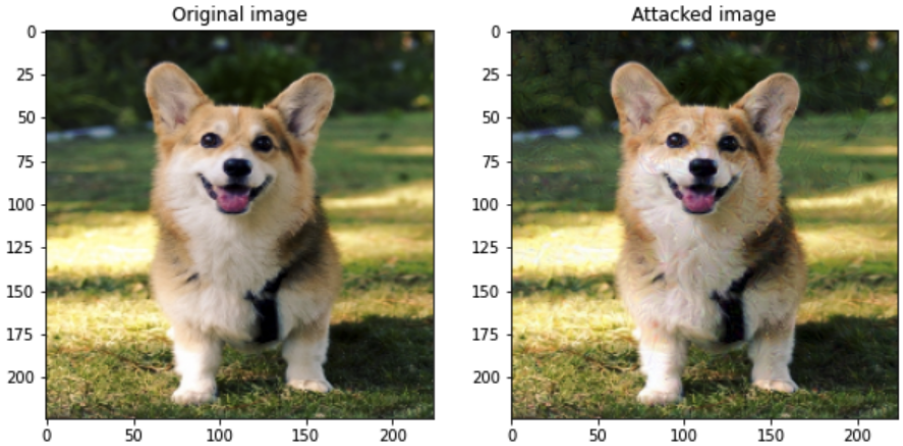

Figure 8: Original Attacked Image of Attack 2 - Tiger Cat (id 282)

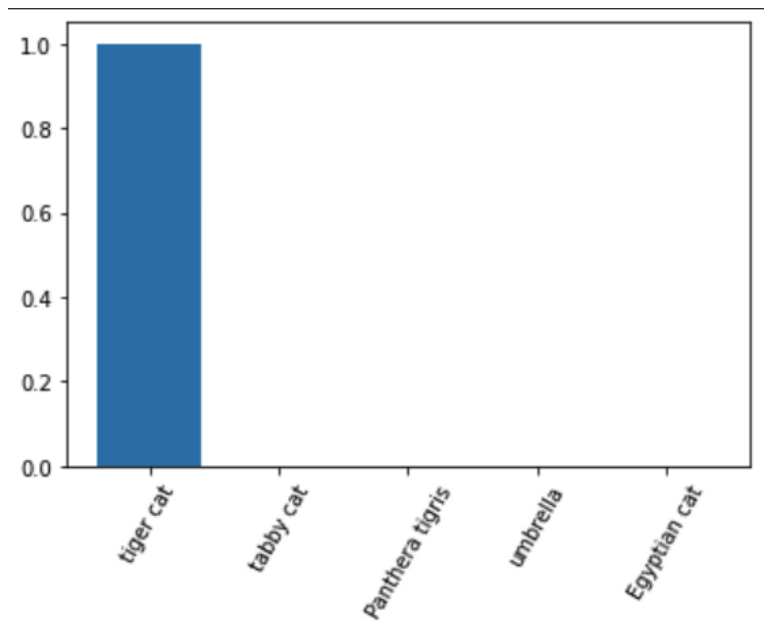

Figure 9: Updated Probability of Attack 2 - Tiger Cat (id 282)

(c)

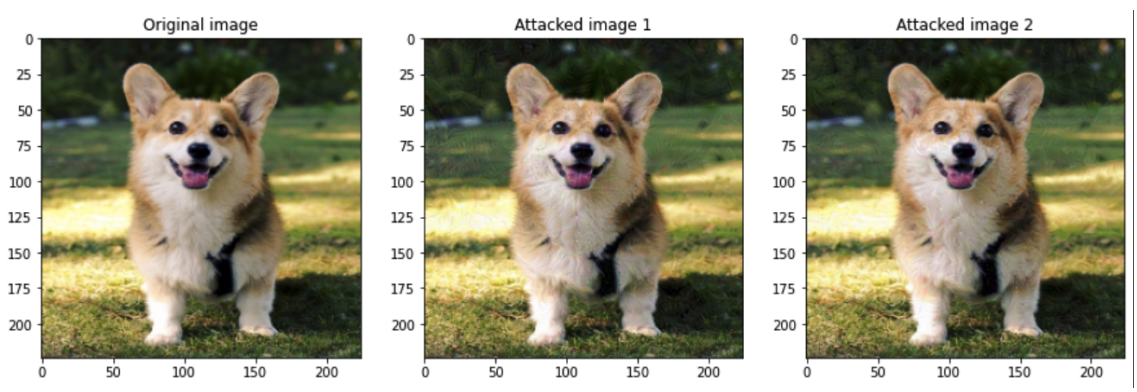

Figure 10: Comparison between attack 1 and attack 2

From human eyes, there's seldom difference between the adversarial images and the original image, except that there are more perturbations, noises and blurring in the background of the adversarial images.

This causes severe security problems for deep neural networks because a small perturbation could cause a drastically difficult prediction outcome. If the deep neural network is not robust to these perturbations, there will be severe privacy and security concerns. For example, we could probably add perturbation to biometrics information such as one's face image, or finger print, which are used in 2FA, and it could be predicted as a completely new person to fool security system. Hackers with malicious intent could exploit the system.

(e) Top Predicted Class: Tiger Cat

The picture looks more like a tiger cat now, especially in the face of the Corgi, it has more pattern like a tiger cat. Other similar features are kept and unchanged (legs).

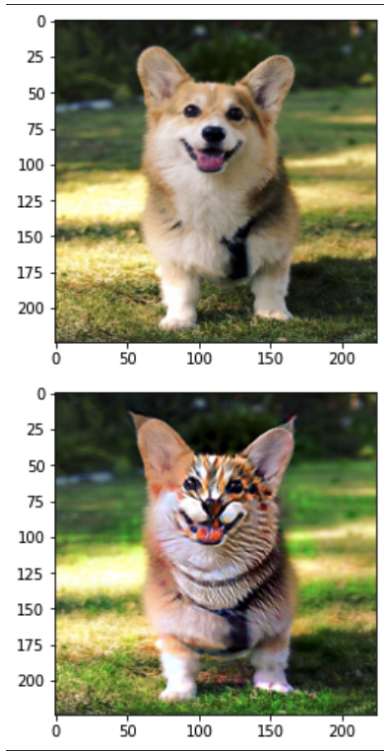

Figure 11: Original Modified Image in Robust Network

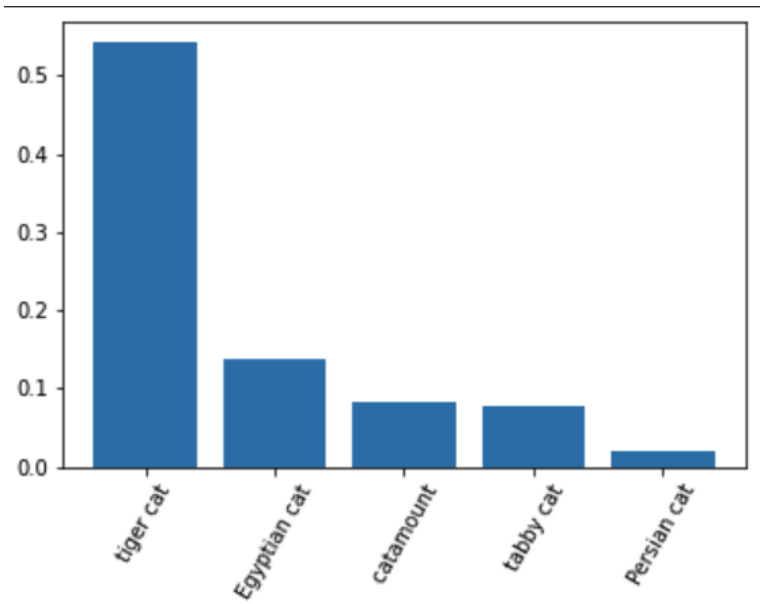

Figure 12: Updated Probability

(f) I picked two random class id to experiment.

The code is as below

```
def perturbation_experiment(class_id):
    # input image is modified to maximize the class_id
    model.eval()
    for param in model.parameters():
        param.requires_grad = False
    initial_image = prepare_image(img, do_normalize=False)
    batch_tensor = initial_image.clone()
    batch_tensor.requires_grad_(True)
    step = StepImage(initial_image, step_size=2, renorm=True, is_normalized=False, eps=30)
    batch_tensor = generate_adversarial_example(model, batch_tensor, class_id)

    # show images
    image = obtain_image(initial_image[0, :], do_normalize=False)
    plt.imshow(image)
    plt.figure()
    image = obtain_image(batch_tensor[0, :], do_normalize=False)
    plt.imshow(image)
    plt.show()

    # updated probability
    output = model(batch_tensor)
    prob = output2prob(output)
    plot_top_classes(prob[0,:].cpu().data.numpy())
    plt.show()

perturbation_experiment(class_id = 77)
perturbation_experiment(class_id = 222)
```

The result looks like something below. For id 77, it is hunting spider, and the image is modified but still have a large portion with corgi. Thus the prediction result isn't very accurate, the highest probability prediction is a sea star instead.

In the id 222, the image is modified with minimal change (only the face of the dog), because Kuvasz is also a kind of dog, and it is similar to Corgi with many shared properties, the final modified image is also similar. And the prediction result is accurate and robust, with highest probability as Kuvasz.

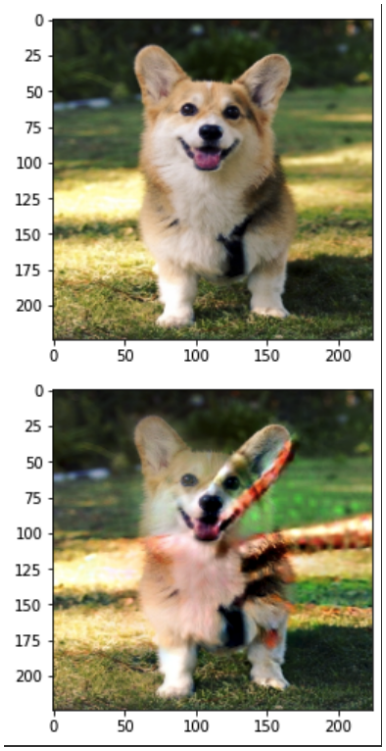

Figure 13: Original Modified Image in Robust Network - id 77

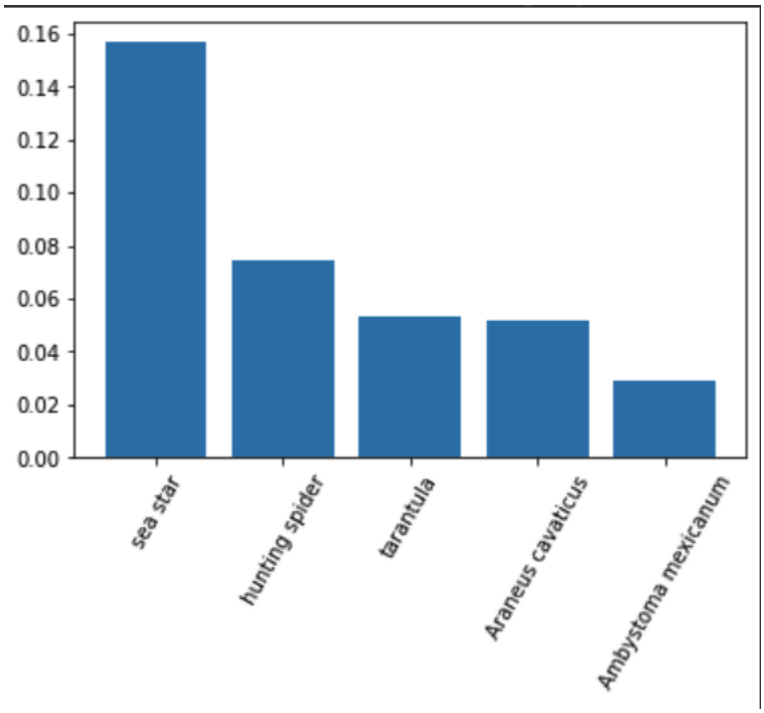

Figure 14: Updated Probability - id 77

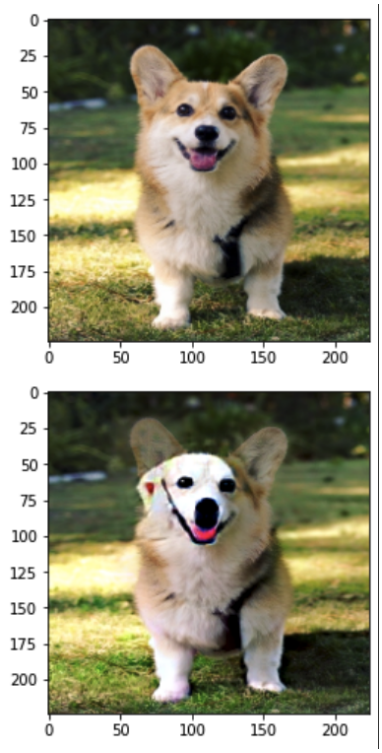

Figure 15: Original Modified Image in Robust Network - id 222

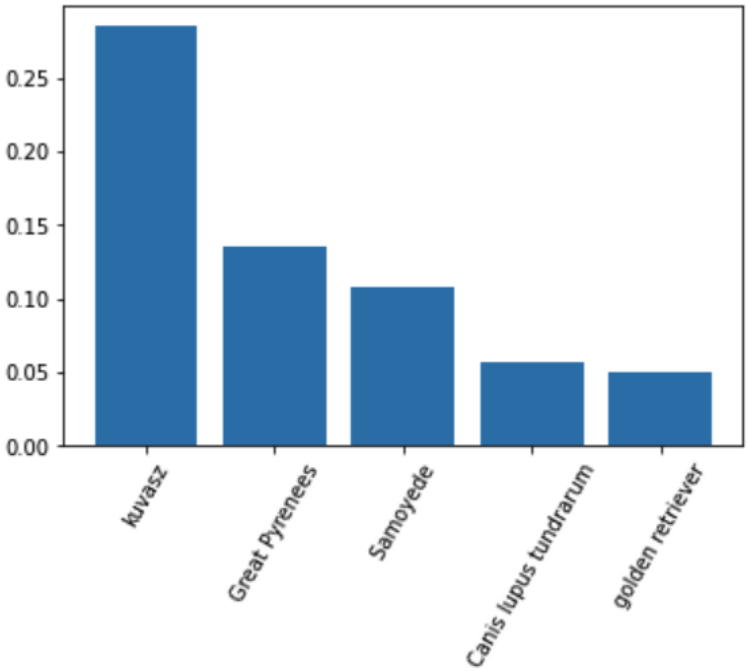

Figure 16: Updated Probability - id 222