Leandra Tejedor
6.8300: Advances in Computer Vision
Pset 3

Problem 1

1a. The solution for forward propagation of a convolutional layer is to loop through x_{in}, applying a W (of that associated part of the kernel) equation to size k pixels to create a new layer of $x_{out}[i]$. The equation below assumes that the kernel has already been flipped.

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

For indices outside of the boundary, this equation will use zero padding. In other words, if the index i + j - ((k - 1) / 2) is either less than 0 or greater than or equal to the length of xin, 0 is used for that term in the summation. The summation runs from j=0 to j=k-1.

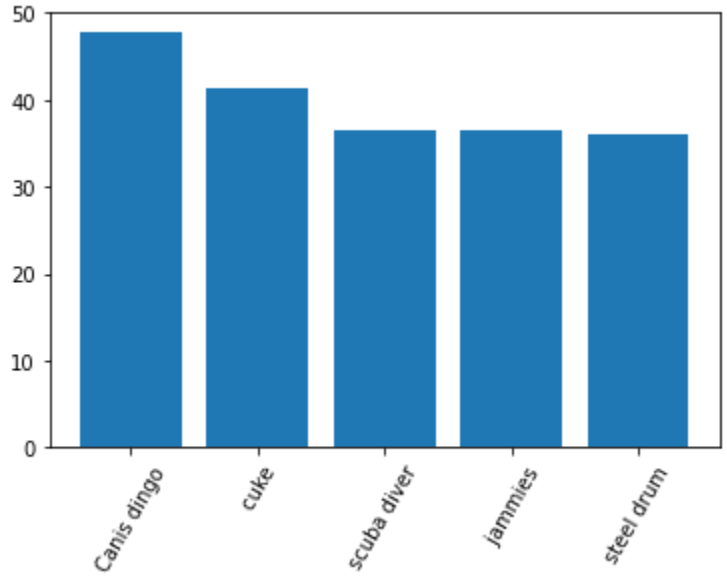
1b. To implement a pooling layer, we loop through x_{in} and compare k values around that point. The max of those k values gets saved in that point of x_{out}. Like the problem above, j starts at 0 and loops k times (from j=0 to j=k-1).

$$x_{out}[i] = \max_{j=0}^{k} (x_{in}[i+j-(k-1)/2])$$

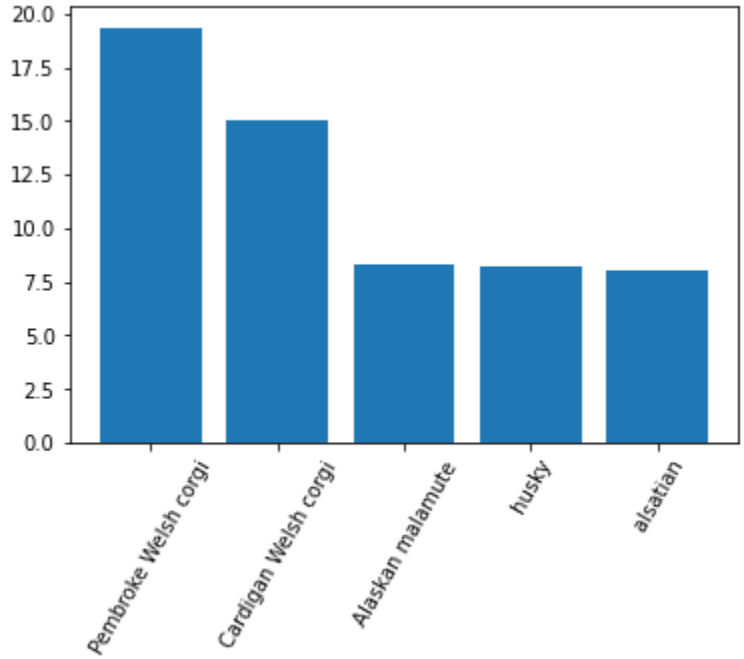
Problem 2

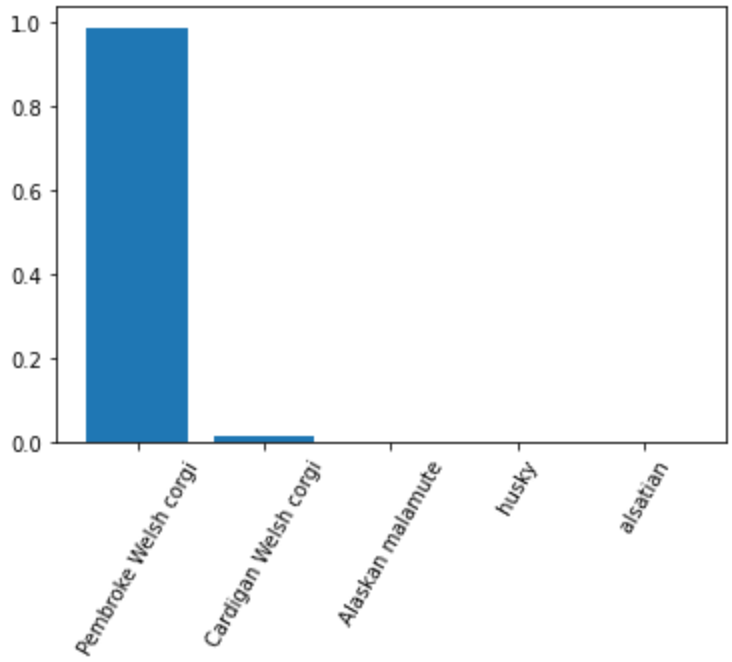
2a. 2048

2b. The top 5 predictions are: Canis dingo, cuke, scuba diver, jammies, and steel drum



2c.





The top 5 predictions are: Pembroke Welsh corgi, Cardigan Welsh corgi, Alaskan malamute, husky, and alsatian

```
[32] def output2prob(output):
       ### TODO1
       # Your code here:
       prob = torch.nn.functional.softmax(output, dim=1)
       ###
       return prob
```

Problem 3

3a. The solution for the backward propagation of a convolutional layer involves looping through the gradient of the output signal (dL/dx_{out}), applying the corresponding kernel weights W[j] to compute the gradient of the input signal ($dL/dx_{in}[i]$). The equation below assumes that the kernel has already been flipped. For indices outside of the boundary, this equation will use zero padding.

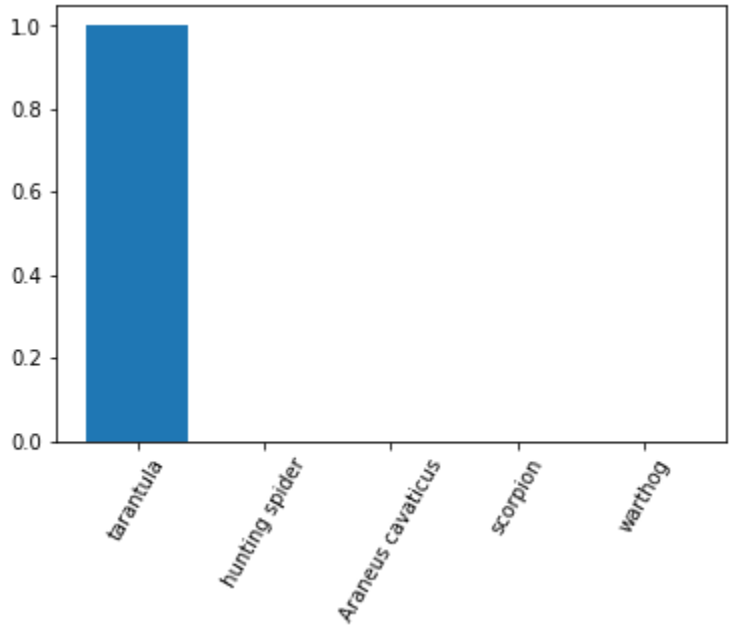
$$dL/dx_{in}[i] = \sum_{j=0}^{k} dL/dx_{out}[i+j-((k-1)/2)] * W[j]$$

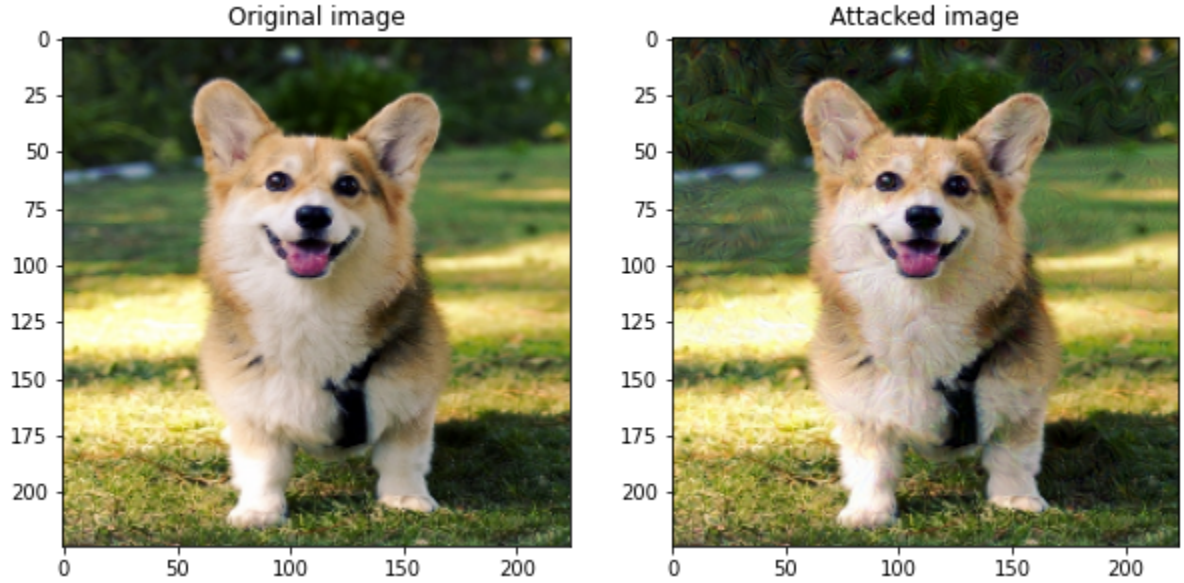
3c. I handled the boundaries using zero padding, so any indices outside of the boundary will be handled as a “0” value.

3e. For this problem, I would have set any indice outside the boundary to -Infinity. Because this is looking for the maximum of each set, I don’t want to set it to 0 because numbers at each point could be negative.

Problem 4

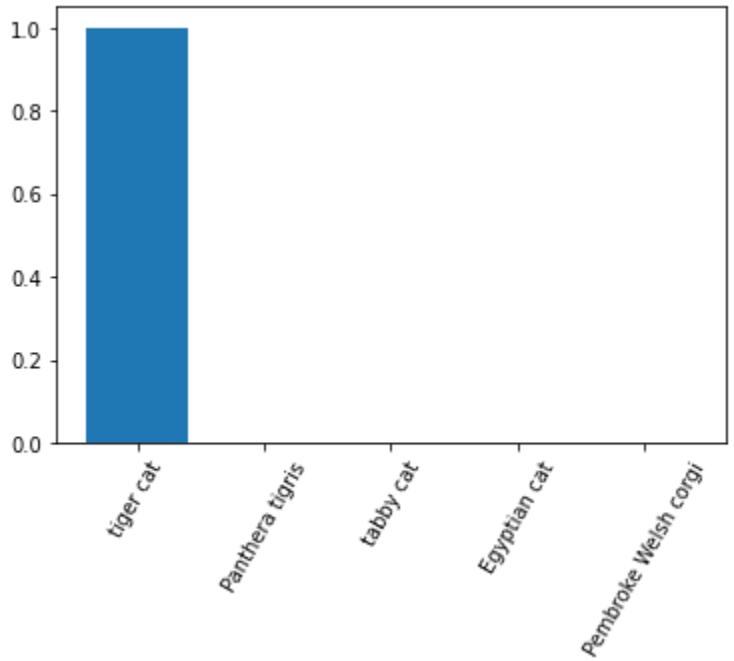
4a.

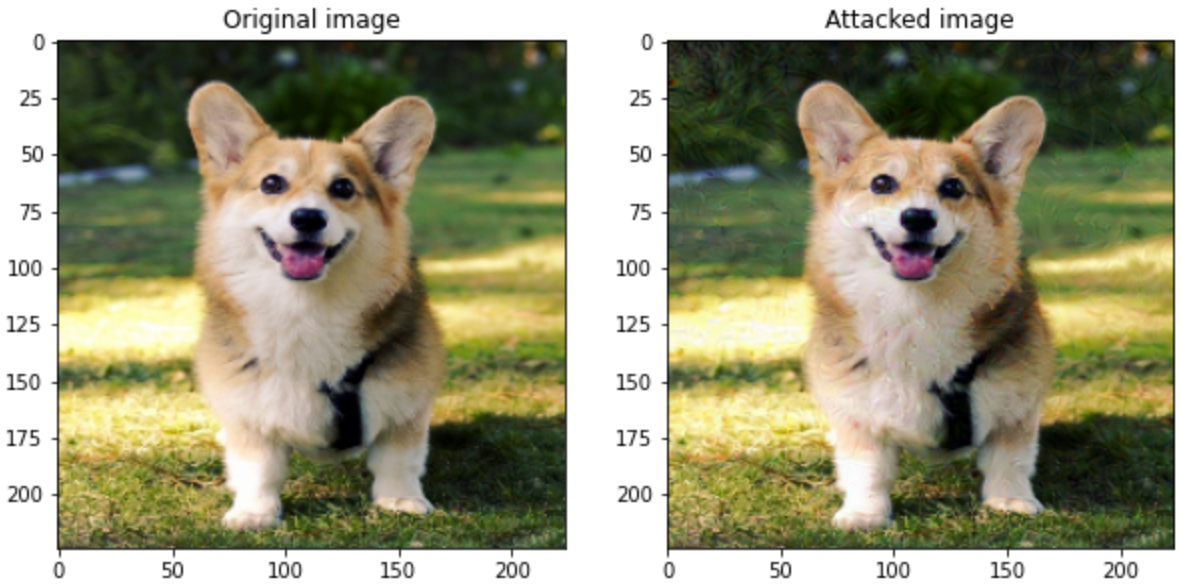

1.0
0.8
0.6
0.4
0.2
0.0
tarantula
hunting spider
Araneus cavaticus
scorpion
warthog


Original image
Attacked image
0
25
50
75
100
125
150
175
200
0
50
100
150
200

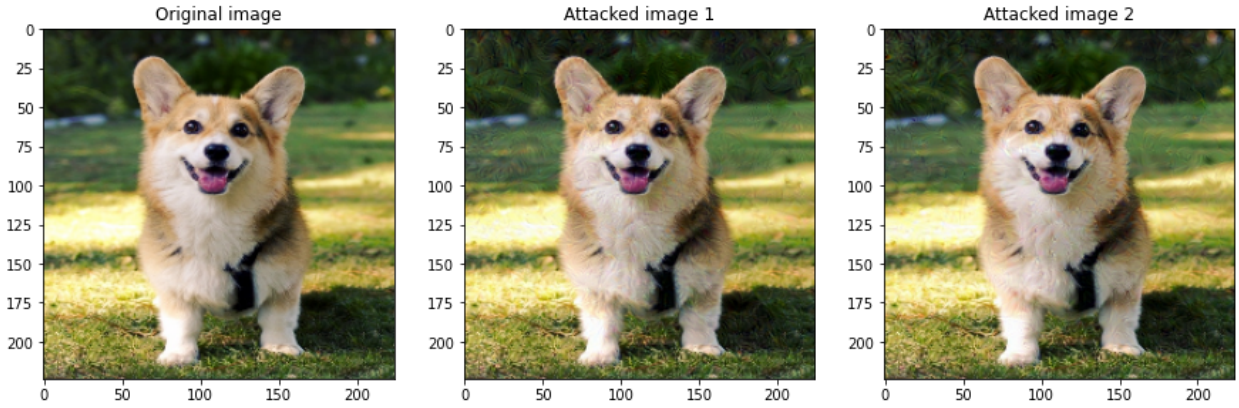
```python
def generate_adversarial_example(model_fn, x, class_id, n_iter=200):
  """
  :param model_fn: a callable that takes an input tensor and returns the model logits.
  :param x: input tensor.
  :param class_id: the id of the target class.
  :return: a tensor for the adversarial example
  """
 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


    logit = model_fn(x)
    loss = torch.nn.CrossEntropyLoss()

    loss_cal = loss(logit, torch.tensor([class_id]).cuda())

    gradient, = torch.autograd.grad(loss_cal, x)
    ###
    x = step.step(x, -1*gradient)
  return x
```

4b.



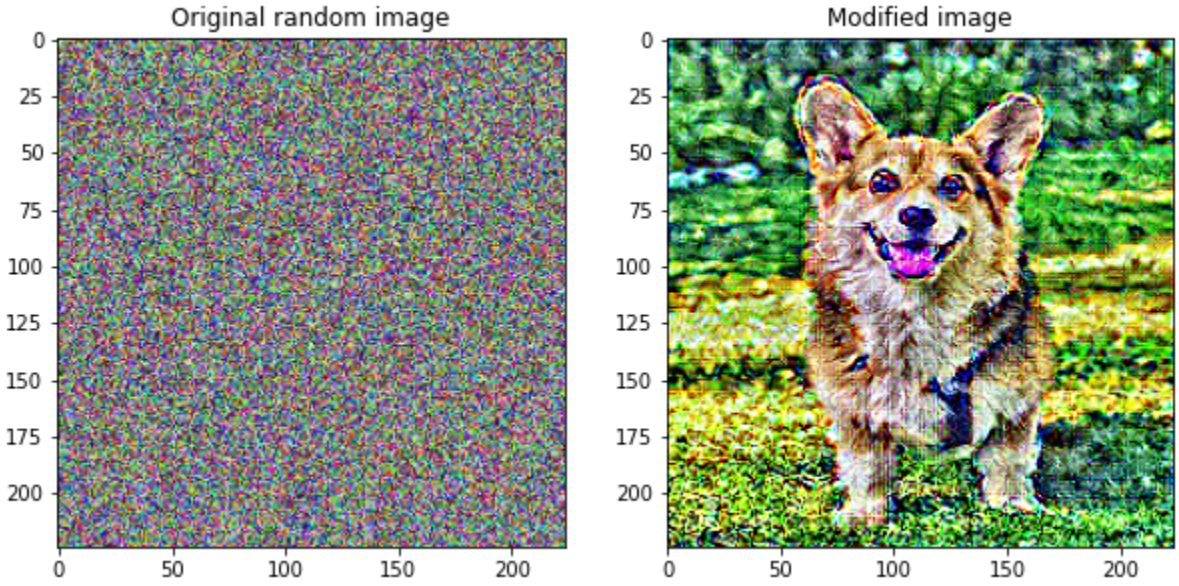


4c.

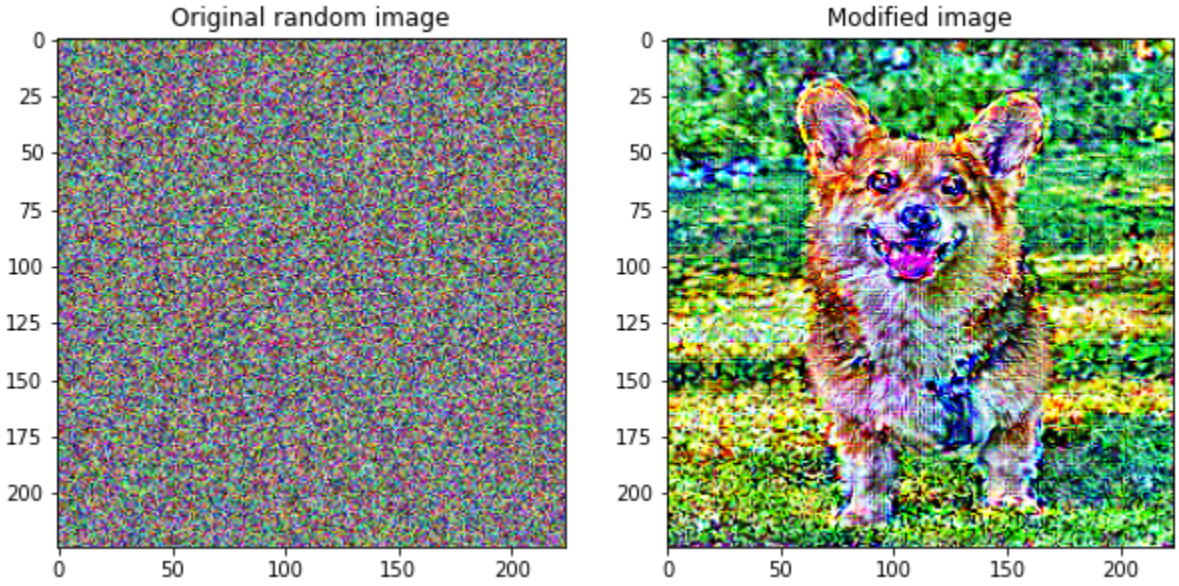


Yes, I can see “swirls” around the image, but only if I’m looking closely and would know to look. Depending on the type of screen that someone was looking at this image, it might be impossible. As AI becomes embedded in everyday security, there may be security problems caused by attacks like this when trying to get into someone’s home or office if their security uses face-id. Another could be if a company is using machine learning to authenticate documents such as IDs, fake documents could get through the system.

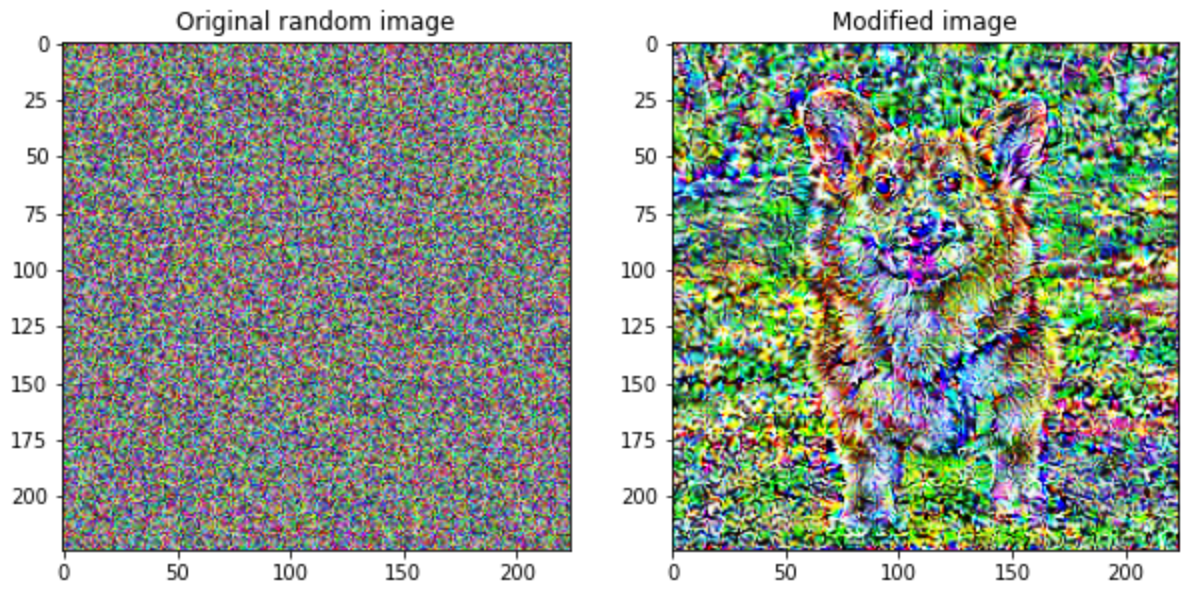
4d.



Layer 0



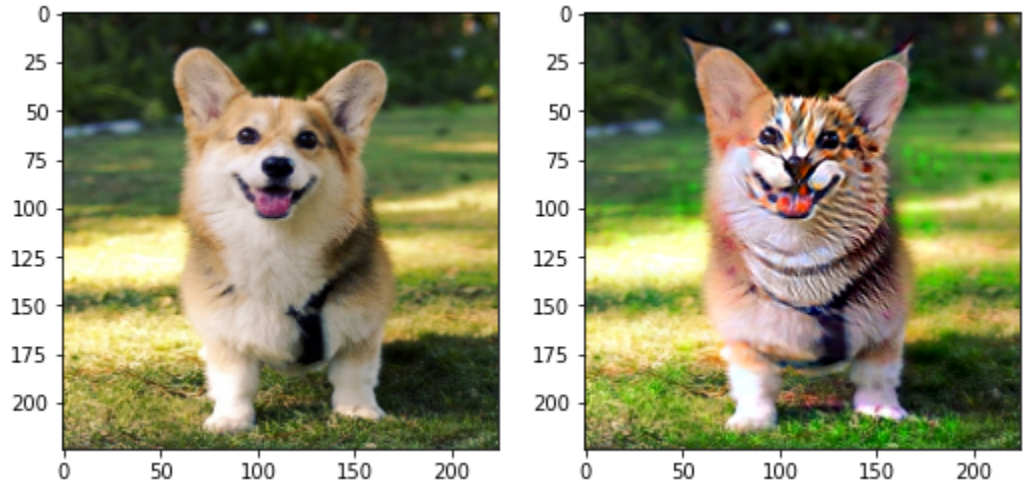
Layer 1

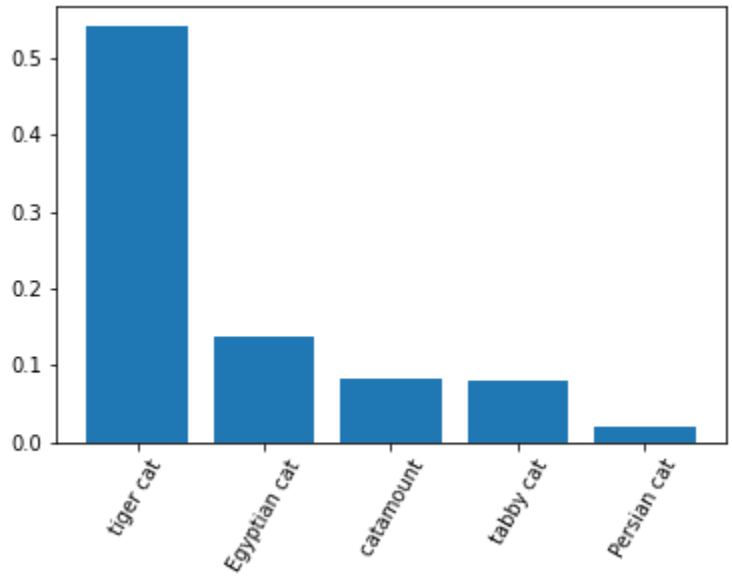


Layer 2

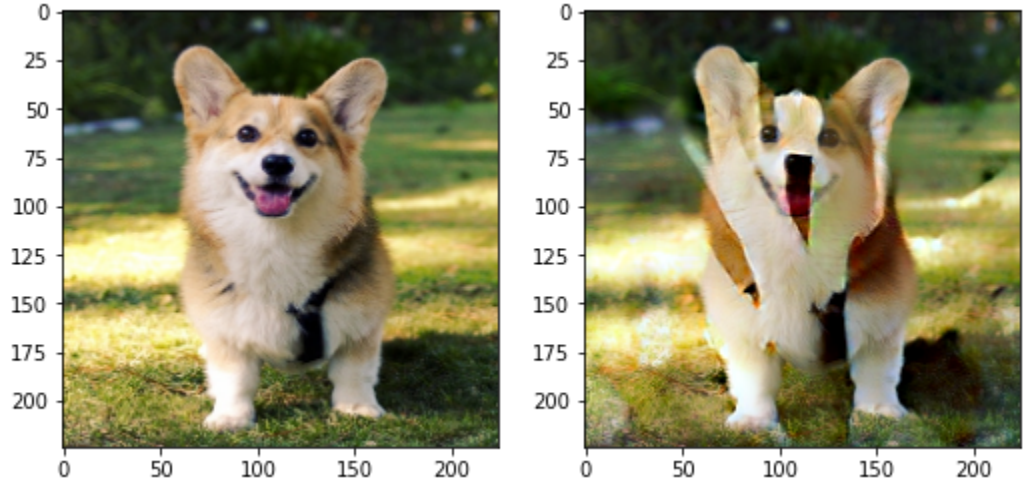
```
### TODO3
# Modify Layer ID (0-3) to select feature from different layers
layer_id = 3
###
model_1 = model_layer(model, layer_id)

# target image and the embedding
target_image = prepare_image(img)
target_feat = model_1(target_image)

for _ in tqdm(range(200)):
  feat = model_1(batch_tensor)
  ### TODO4
  # You should minimize the L2 norm between feat and target_feat
  # Your code here:
  loss_fn = torch.nn.MSELoss()
  loss = loss_fn(torch.norm(feat, p=2, dim=0), torch.norm(target_feat, p=2, dim=0))
  ###
  gradient, = torch.autograd.grad(loss, batch_tensor)
  batch_tensor = step.step(batch_tensor, -gradient)
```

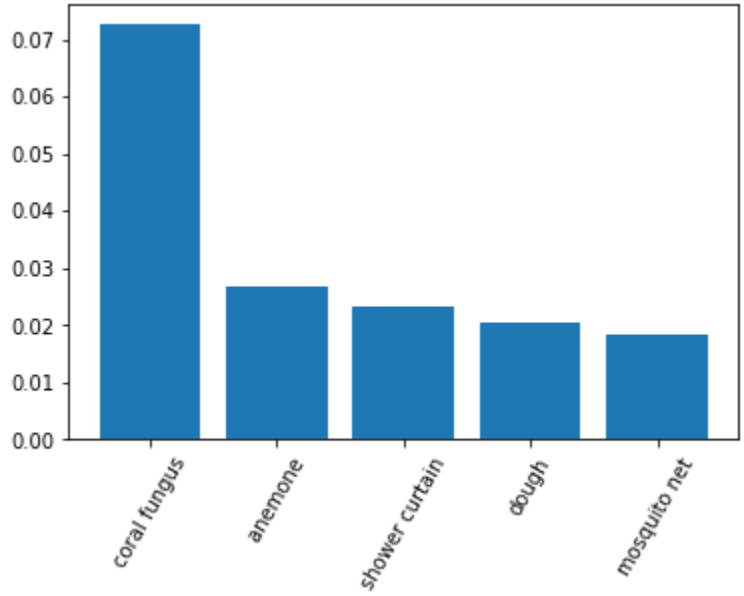
4e.


0
25
50
75
100
125
150
175
200
0
50
100
150
200

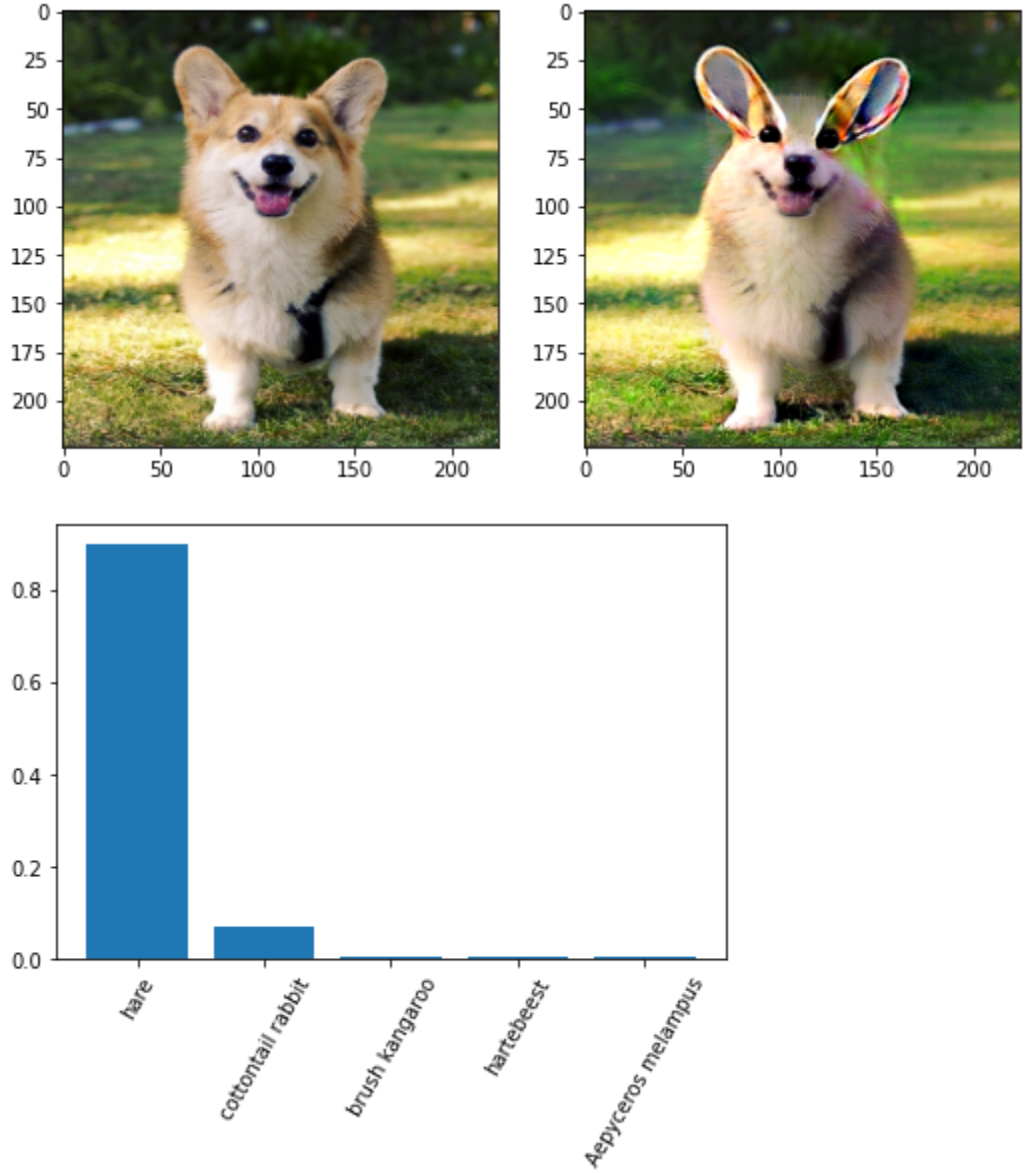

0.5
0.4
0.3
0.2
0.1
0.0
tiger cat
Egyptian cat
catamount
tabby cat
Persian cat

Class 991 Coral Fungus



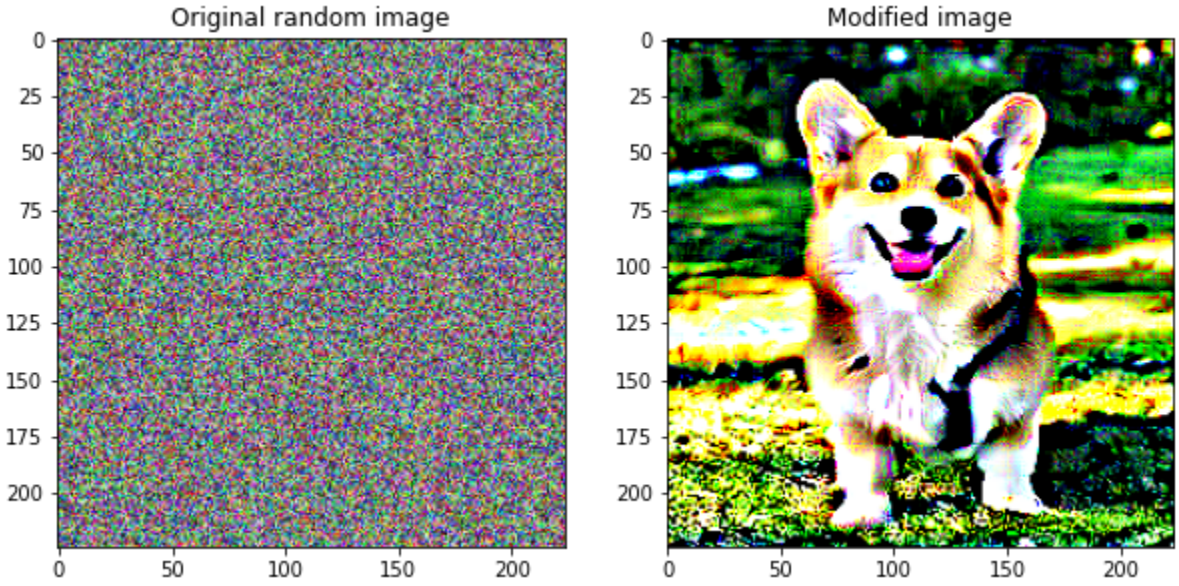


Class 331 - Hare

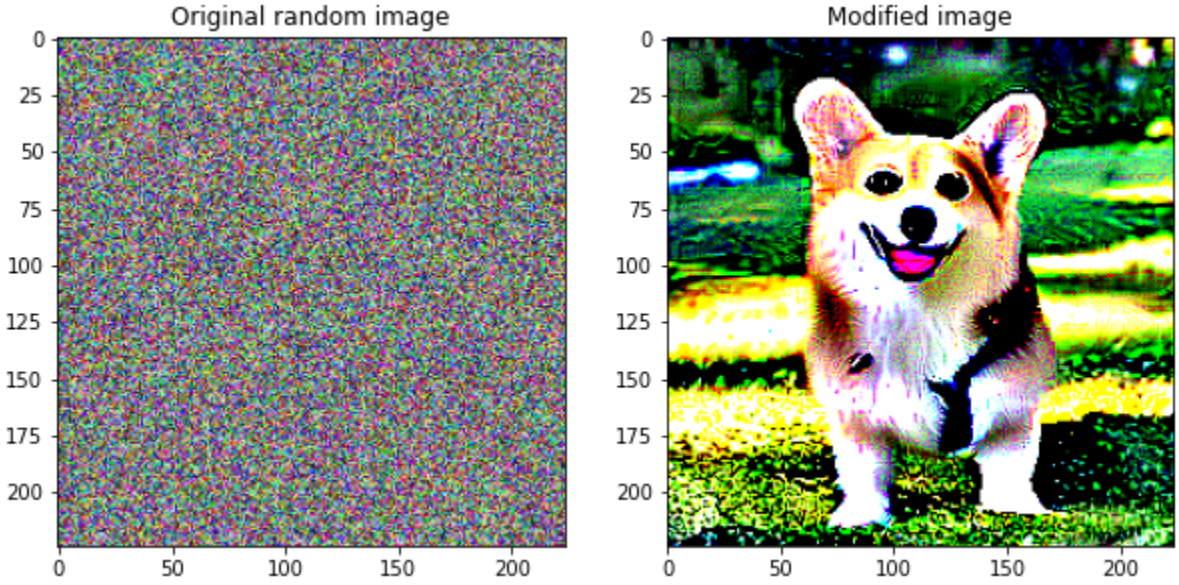


The attacked images look significantly different. Interestingly, coral modifies the colors in a way that still reads “dog” to my human brain, while the other two that morph it into dog-adjacent animals have very clear differences.

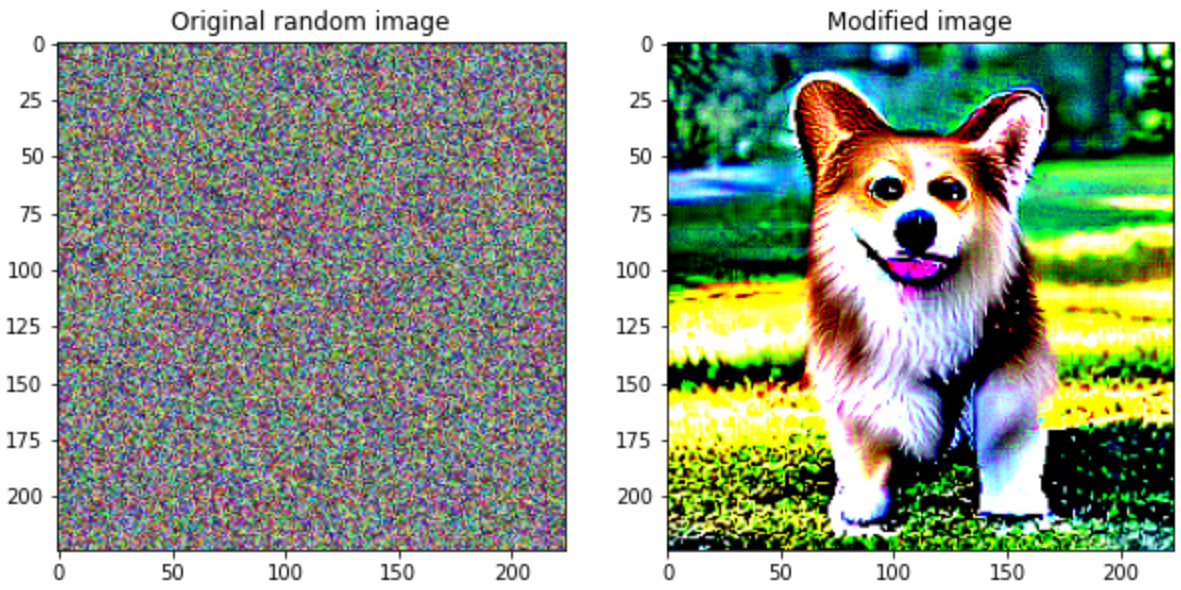
4f.



Layer 0



Layer 1



Layer 2

```
### TODO5
# Modify Layer ID (0-3) to select feature from different layers
layer_id = 1
###
model_l = model_layer(model, layer_id)

# target image
target_image = prepare_image(img)
target_logit = model_l(target_image)

for _ in tqdm(range(200)):
  logit = model_l(batch_tensor)
  ### TODO6
  # You should minimize the L2 norm between logit and target_logit
  # Your code here:
  loss_fn = torch.nn.MSELoss()
  loss = loss_fn(torch.norm(logit, p=2, dim=0), torch.norm(target_logit, p=2, dim=0))
  ###
  gradient, = torch.autograd.grad(loss, batch_tensor)
  batch_tensor = step.step(batch_tensor, -gradient)
```

The images with this model seem to warp, whereas before they lost their form and “dissolved” into the noise. I think they do look more similar to the original corgi image here.