6.869 PSet 4

Nada Hussein

March 2021

Problem 1

a) if x_{in} and x_{out} are both length N, and the kernel W has odd size k, we know that the relationship between the input and output is governed by:

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

I have assumed that stride is 1 here. We can express this as a convolution on the range of W:

$$x_{out}[n] = \sum_{m=0}^{m=k-1} W[m]x_{in}[n-m]$$

This will generate an x_{out} of length N - k + 1, since this will only become a valid operation that uses all elements of the filter starting at index k/2 and ending at index -k/2.. However, if we want our output to have length N, we can use zero-padding of (k-1)/2 on each side to bring the length back up to N.
b) If we have max pooling with stride 1, and we know that the input has size N and the output has a different size M, we know that the maxpool must be computed over k = N-M+1 indices at a time. We also assume 0 padding. In this way, any index of x_{out} will be the maximum value of x_{in} between index n and index n+k as follows:

$$x_{out}[n] = \frac{1}{k} \max_{m \in [n, n+k]} x_{in}[m]$$

This will result in x_{out} having length N-k+1 = N-(N-M+1)+1 = M, where M is the length of x_{out} that we anticipated.

Problem 2

a) The last layer has 2048 inputs, as we can see from the line:
Linear(in_features=2048, out_features=1000, bias=True)
b) After running the Corgi image through the randomly initialized network, I got cuirass, water tower, coffeepot, shoji, and whiptail lizard as predictions. The chart is shown below:

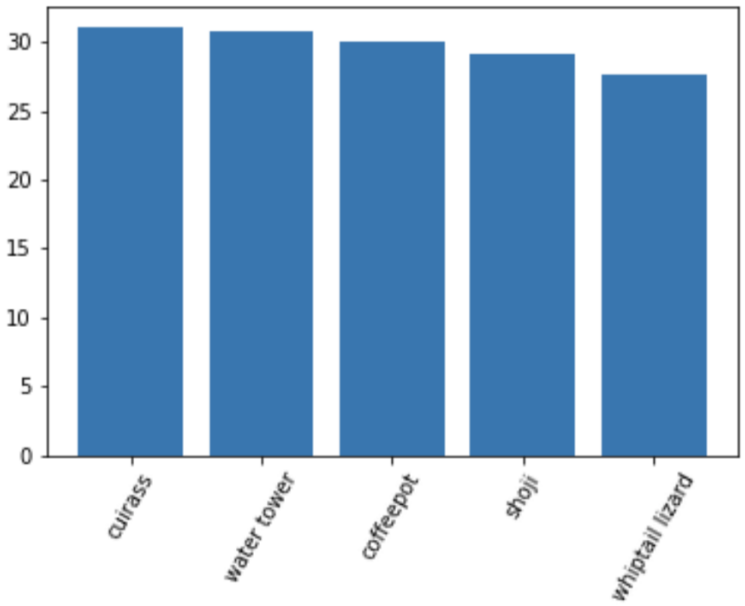

Figure 1: Corgi Classification Random Initialized Network

c) I ended up with the following predictions after rerunning the model with pre-trained weights:

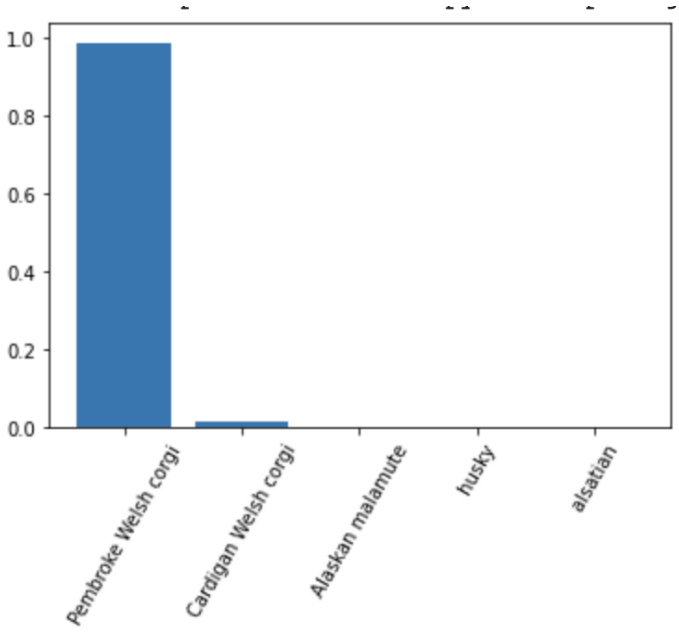

Figure 2: Corgi Classification

Here you can see the top predictions were Pembroke Welsh corgi and Cardigan welsh corgi, but the top prediction, Pembroke Welsh, had about 99% probability.

Problem 3

a) We start with our basic equation for backward propagation:

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

We are given $\frac{\partial C}{\partial x_{out}}$, so we simply need to find $\frac{\partial x_{out}}{\partial x_{in}}$. We can define the relationship between x_{in} and x_{out} as follows:

$$x_{out} = Mx_{in} = \begin{bmatrix} W[0] & \dots & W[-\lceil k/2 \rceil] & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & W[\lceil k/2 \rceil] & \dots & W[0] \end{bmatrix} x_{in}$$

Using this matrix M, we can then find that $\frac{\partial x_{out}}{\partial x_{in}} = M$. Thus we end up with:

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

b) Once we have the equation from part a, we now need $\frac{\partial C}{\partial W}$. We can use the chain rule as follows:

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

We are given We know that this will $\frac{\partial C}{\partial x_{out}}$, so we just need to find $\frac{\partial x_{out}}{\partial W}$, which take the values of x_{in} as derivatives based on our convolution equation. We know the relationship for this derivative looks like:

$$\frac{\partial x_{out}[i]}{\partial W[j]} = x_{in}[i-j]$$

I have indexed my W from -k/2 to k/2, so we get the following: Thus we end up with:

$$\frac{\partial x_{out}}{\partial W} = \begin{bmatrix} x_{in}[\lceil k/2 \rceil] & \dots & x_{in}[0] & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & x_{in}[-\lceil k/2 \rceil] & \dots & x_{in}[N-1-\lceil k/2 \rceil] \end{bmatrix}$$

This creates an N x k matrix, which matches the dimensions we would expect. Thus leads to the update rule:

$$W^{i+1} = W^i - \eta \frac{\partial C}{\partial W} = W^i - \eta \frac{\partial C}{\partial x_{out}} \frac{\partial x_{out}}{\partial W}$$

with partials as defined above.

c) To handle boundaries, I decided to do zero padding. This way, we end up maintaining the shape that we want, but 0s ensure that we will not be changing the actual calculation in each layer as the 0 values will not contribute to the convolution.

d) We start with the following:

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

To find $\frac{\partial x_{out}}{\partial x_{in}}$, we need to find a relationship. We know that each time the kernel is applied, we will end up with mostly 0s, and any value that was maximum in the area of the input that we look at will remain as is. This means the partial derivative will be 1 if we kept the value, and 0 if we did not. We can represent this as follows:

$$x_{out} = Px_{in} \text{where} P[i,j] = \begin{cases} 1 & x_{in}[j] = x_{out}[i], 0 \leq i-j \leq k-1 \\ 0 & \text{else} \end{cases}$$

Here we have asserted it is only 1 if the value at $x_{in}[j]$ is the max, and that index j was included in the ith value of the output. This leads to our backwards propagation rule:

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

with P defined above.
e) For max pooling, I handled boundaries by removing boundary values in x_{out}. We cannot zero pad here, because this could impact the result if the values of x_{in} are negative. Thus, we simply do not include the boundaries in our final result.

Problem 4

a)

To create the adversarial examples, I used the following code:

```
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
    logit = model_fn(x) # Your code here
    lossFunc = torch.nn.CrossEntropyLoss()
    target = logit[0,class_id]
    loss = lossFunc(logit, torch.tensor(class_id).reshape(1,).to('cuda')) # Your code here
    gradient = torch.autograd.grad(loss, x)[0]# Your code here
    ###
    x = step.step(x, -gradient)
  return x
```

Figure 3: Adversarial Code

Below you can see the resulting image using the tarantula as an adversarial example, which leads to a tarantula classification:

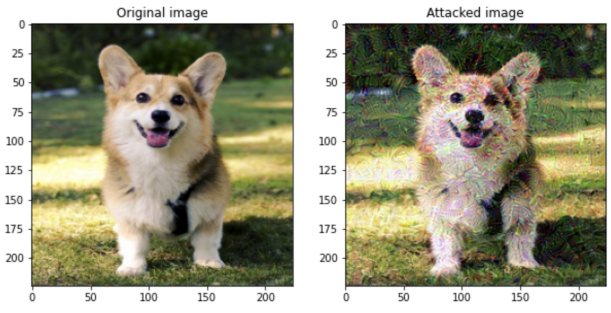

Figure 4: Tarantula Adversarial Image

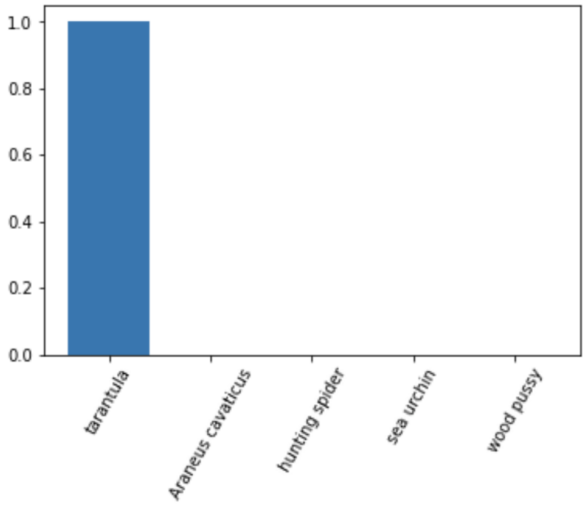

Figure 5: Tarantula Adversarial Image Classification

b) Below you can see the resulting image using the tiger cat as an adversarial example, which results in a tiger cat classification:

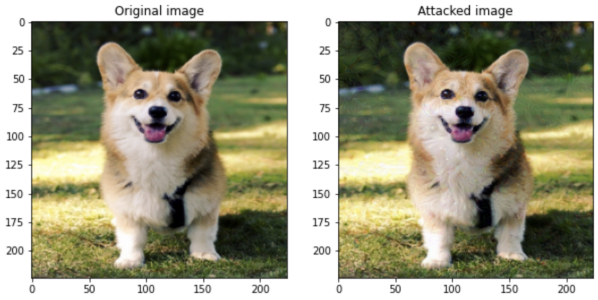

Figure 6: Tiger Cat Adversarial Image

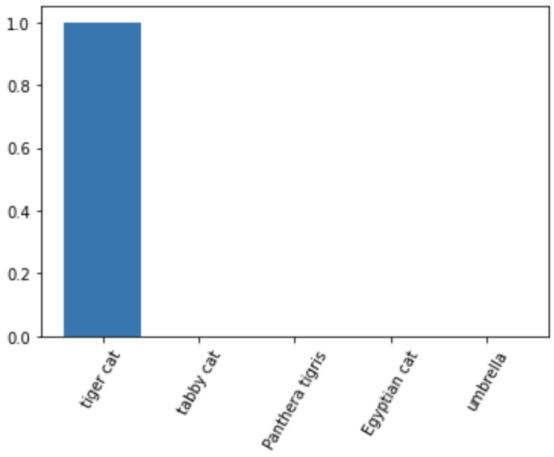

Figure 7: Tiger Cat Adversarial Image Classification

c) The tarantula adversarial example distorts the corgi much more than the tiger cat. This happens because the tiger cat is much more visually similar to a corgi than the tarantula is, so it needs to change less visibly for the image to be classified as a tiger cat.
d) To implement the L2 norm maximizing, I used the following line of code:
loss = torch.norm(logit)
Using this, I've shown below a few examples of this run on different layers:

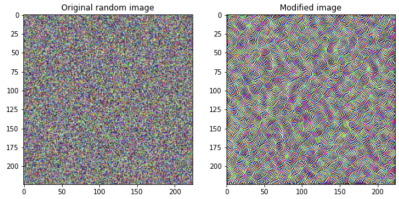

Figure 8: L2 Norm Layer 0

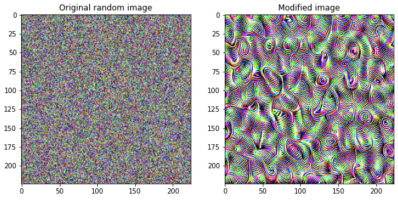

Figure 9: L2 Norm Layer 1

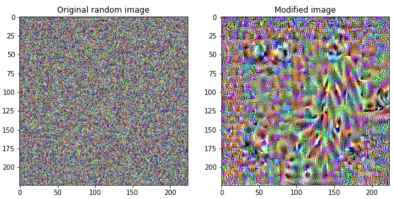

Figure 10: L2 Norm Layer 2

e) After running the robustly trained network, the image appears as follows:

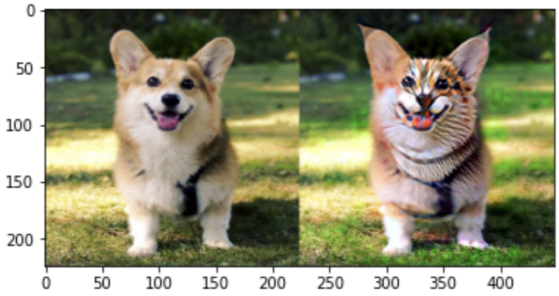

Figure 11: Robust Network Result

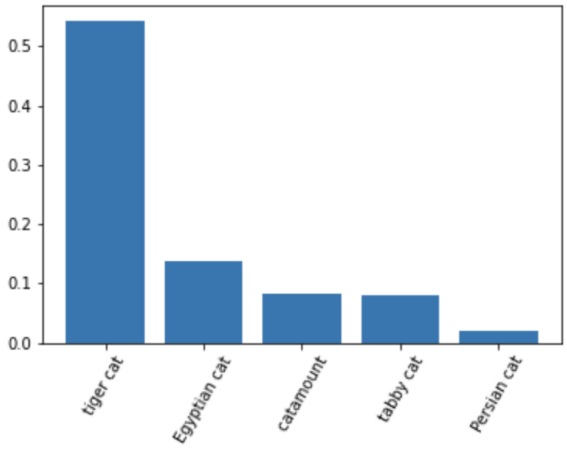

Figure 12: Robust Classification

Here we can see that the image takes on the features of a tiger cat in order to be classified as such.
f) Below are a few examples of the robust network modifying the image into other classes:

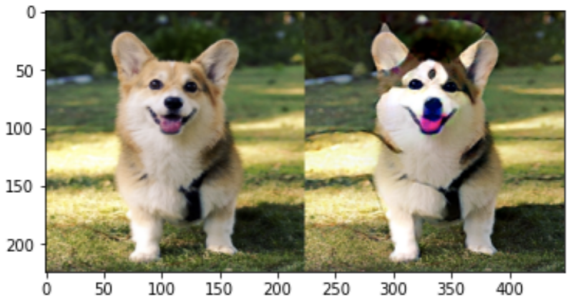

Figure 13: Robust Network Husky

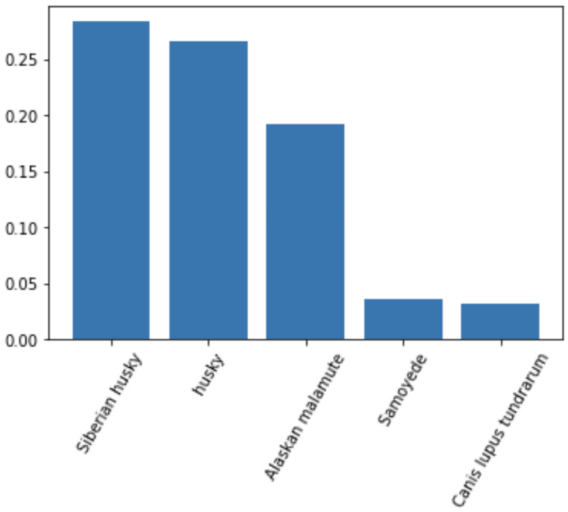

Figure 14: Robust Husky Classification

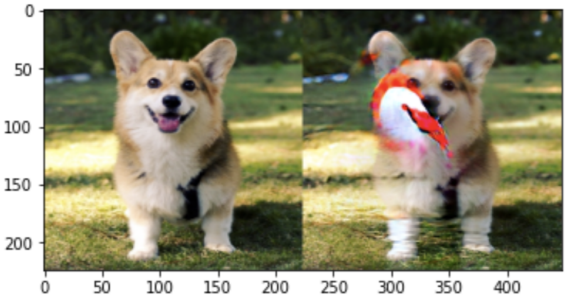

Figure 15: Robust Network Flamingo

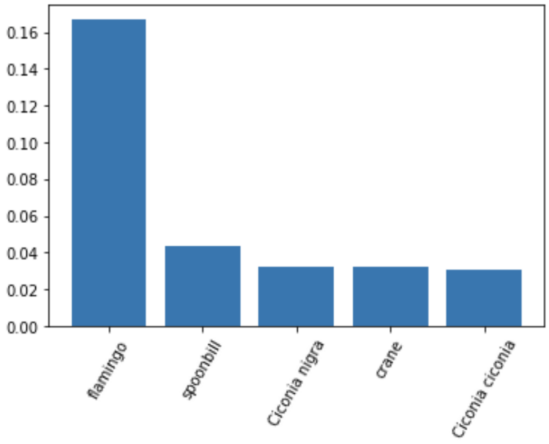

Figure 16: Robust Flamingo Classification

Here we see that changing the corgi into a husky classification actually resulted in changing the coat of the dog into a husky pattern. However, the flamingo ended up causing strange flamingo artifacts overlaid on the corgi since they were not as similar.