1 Median Filtering

Recall that a convolutional kernel in image contexts involves calculating the sum of the elementwise products of the kernel with a window of pixels in the image, then sliding the kernel across the image, repeating the process for every pixel in the image.

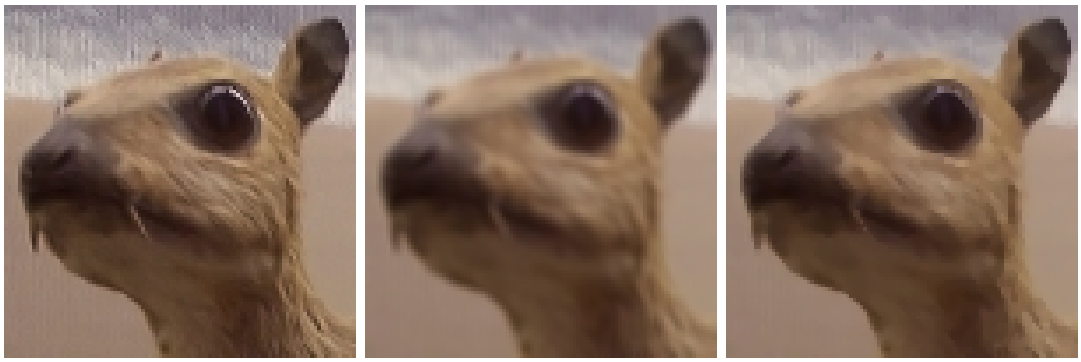
(a) Consider the following 3x3 kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

This is often referred to as a "box-blur" kernel. Why does it have a blurring effect?

(b) Now consider a related technique, known as *median filtering*. Here, for a given window of pixels, the center pixel is replaced with the median of the pixels in the window. This process is then repeated across the entire image, just as is done with convolution.

For your curiosity, depicted below are the results of applying 3x3 filters to an image (original left, box-blur center, median right):



Can you think of a scenario where median filtering would be preferable to box-blur filtering? What about the other way around?

(c) For ConvNets, are median filters similar to convolutional layers? Are they still useful? Why or why not?

(d) Consider the infinite zero matrix

$$\mathbf{0} = \begin{bmatrix} \ddots & \vdots & \vdots & \iddots \\ \cdots & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \cdots \\ \iddots & \vdots & \vdots & \ddots \end{bmatrix},$$

which, for this problem, represents a bitmap of constant color. Now suppose each pixel in the image is independently at random flipped to 1 with probability $p < 0.2$, representing random noise. What is the expected result of applying a 3×3 box-blur filter to this newly noised image? What about a 3×3 median filter? Assume the stride length is at least 3 (so there is no overlap).

Do your answers to (b) and (c) now change?

(e) Let $K \times K$ denote filter size, P padding size, S stride length, and suppose we are operating on an image of size $H \times W$. What is the runtime complexity of convolution? How about median filtering? Assume no results are cached across different pixels.

Hint: You can find the median of n numbers in $O(n)$ time.

2 Weighted Cross-Entropy

Recall that cross-entropy loss is given by

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}),$$

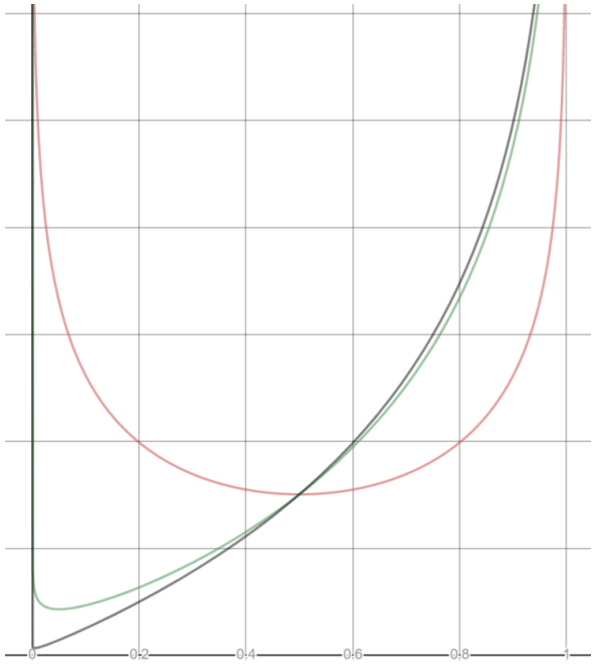
where y is the ground truth label and $\hat{y}$ is the predicted probability that the label is 1.

(a) Suppose p is the ground-truth probability of a positive sample. Find the value of $\hat{y}$ that minimizes the expected loss

$$\mathbf{E}_{y \sim \text{Bernoulli}(p)} \left[\mathcal{L}(y, \hat{y}) \right]$$

by taking the derivative with respect to $\hat{y}$ and setting it to zero.

(b) Now, consider how the model is affected by p. Below is the graph of $\mathbf{E}[\mathcal{L}]$ as a function of $\hat{y}$ for $p = 0.5$, $p = 0.05$, and $p = 0.005$. From this, comment on what problems may arise when using binary cross-entropy loss (you do not need to show these rigorously).



(c) Suppose we want to modify the loss function to be more robust to unbalanced distributions. One way to do this is to introduce a hyperparameter α, which we can use to penalize incorrect predictions more heavily for the minority class (or equivalently, less heavily for the majority class). This is called *weighted cross-entropy* and is given by

$$\mathcal{L}_\alpha(y, \hat{y}) := -\alpha y \log \hat{y} - (1 - y) \log(1 - \hat{y}).$$

(i) Find the value of $\hat{y}$ that minimizes the expectation of this loss. Remember that the ground-truth probability of a positive sample is p, and $y \sim \text{Bernoulli}(p)$.

(ii) Use the value of $\hat{y}$ from part (i) to find α in terms of p such that, when averaged over sufficiently many samples, both the minority and majority classes contribute to the total loss equally.

Note: This derivation is quite difficult, but the final answer should be relatively simple. Feel free to use a symbolic or graphing calculator.