CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 5 (Algorithms) Problem 1

Read: Algorithms

Solution: Complete ■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 5 (Algorithms) Problem 2

Solution: $6n+2$ comparisons ■

Proof: First we can find M_c, the number of pools to be computed for this example with trivial row space.

$$L_c = 2n+1 \qquad F_c = 3 \qquad S_c = 2 \tag{1}$$

so

$$\begin{aligned} M_c &= \left\lfloor \frac{L_c - F_c}{S_c} \right\rfloor + 1 && (2)\\ &= \left\lfloor \frac{2n+1-3}{2} \right\rfloor + 1 && (3)\\ &= \left\lfloor \frac{2n-2}{2} \right\rfloor + 1 && (4)\\ &= \lfloor n-1 \rfloor + 1 && (5)\\ M_c &= n && (6) \end{aligned}$$

Now that we know M_c we need a way of determining how many comparisons are needed to compute each M_c. Our picture of the max pooling operation across an input feature map for the case

$$2n+1 = 21 \qquad n = 10 \qquad F_r = F_c = 3 \qquad S_c = 2 \tag{7}$$

looks like

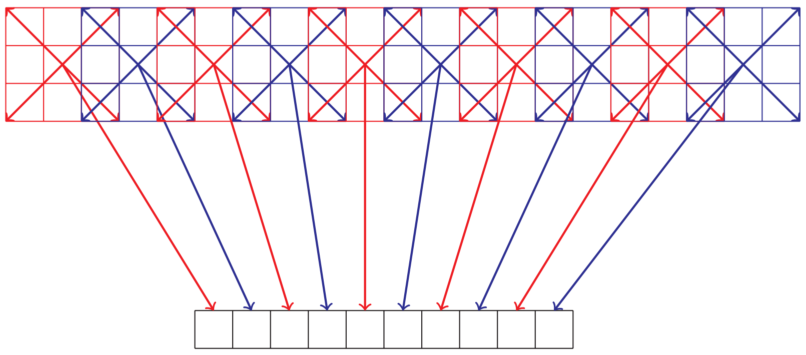
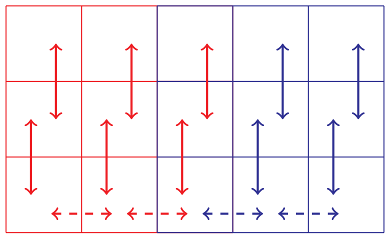


Figure 1. $F_r = F_c = 3$, $2n+1 = 21$, $M_c = 10$

If we implemented a naive comparison strategy, we could compare each item in a pool once, leading to $F_r * F_c - 1$ comparisons for each of M_c pools. However, we know that a solution based on sequential merge sort can yield a maximum in $O(N \log_2(N))$ comparisons.

Suppose that we find a pool's maximum by comparing each of F_r items in a given column. Then we can compare each column maximum across F_r rows to find the pool maximum. In this case we are dividing into three columns, "sorting" out the maximum, and then merging across columns. This also offers us an advantage for the given kernel size, namely that one column's maximum can be reused where strides overlap. If we store the maximum for this column between pools we can eliminate the two comparisons that would otherwise be repeated, cutting our comparisons per pool down to 6 for the non-edge pools. This is visualized as follows, with **dashed lines** indicating comparisons between column maximums.



So we know that we will have 8 comparisons for one of the edge pools, and 6 comparisons for each of the $M_c - 1$ pools.

$$\begin{aligned} \text{Comparisons} &= 8 + (M_c - 1) * 6 && (8) \\ &= 8 + (n-1) * 6 && (9) \\ &= 8 + 6n - 6 && (10) \\ &= 6n + 2 && (11) \\ & && (12) \end{aligned}$$

From the above picture we can count $10 + 4 = 14$ comparisons for the case $2n + 1 = 5$. So for $n = 2$ we will have $12 + 2 = 14$ comparisons. Note that this result of $6n + 2$ comparisons represents a significant savings over the $8n$ comparisons in the naive case. Also note that for a larger kernel size the algorithm would resemble mergesort with logarithmic comparison asymptotes across filter rows and columns. We would potentially lose the overlap between strides for other kernel sizes and strides.

$\square$

CS 6301.503 Spring 2019 — Scott C. Waggener (scw180000)

Homework 5 (Algorithms) Problem 3

Solution: $m(6n+2), n > m$ ■

Proof: We can infer M_r from the result of the previous problem. The pooling operation can be visualized as

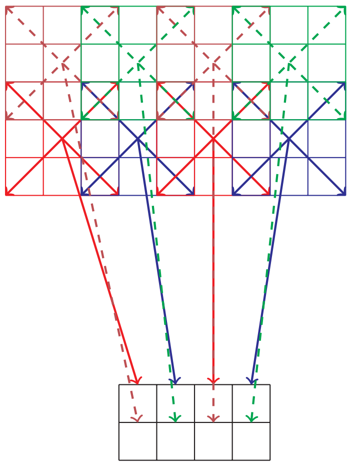
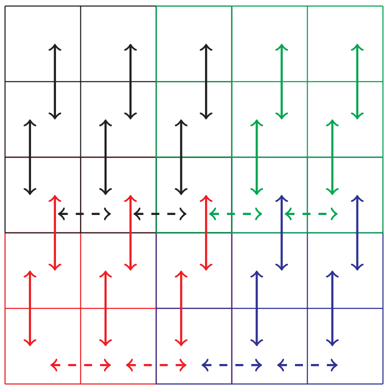


Figure 2. Ugly picture of the pooling operation for non-trivial rows

From this unpleasant picture we can see that the construction from the trivial row case is still viable, however we would see no savings from the overlapping of row maximums. Our comparisons would be as follows, again with **dashed** lines representing comparisons between maximums



In this case, our minimum number of comparisons is achieved by exploiting the stride overlap along the longer of m, n yielding the result

$$\text{Comparisons} = m \cdot (6n+2) \qquad n > m \tag{13}$$

□