Explanation of Algorithms for Convolution

Explanation of Algorithm 1:

Algorithm 1 Pseudocode for computation of convolution

```
1:  for co = 1:chl_o do
2:      for r = 1:r_o do
3:          for c = 1:c_o do
4:              tmp ← 0
5:              for r' = 1:r_k do
6:                  for c' = 1:c_k do
7:                      for ci = 1:chl_i do
8:                          tmp ← tmp + input[r + r' − 1, c + c' − 1, ci] ∗ kernel[co, r', c', ci]
9:                      end for
10:                 end for
11:             end for
12:             output[r, c, co] ← tmp
13:         end for
14:     end for
15: end for
```

This is the dot product of kernel (r_k * c_k * chl_i) with input [(r : r + r_k), (c : c + c_k), chl_i]

* for each output channel *CO*, we are using different kernel

- The lines from 5 - 11 compute a dot product between kernel and a part of input to produce a single value of output
- This single value of output is stored in line 12
- This process of dot product is first calculated for an entire row
- Then we increment to next row and calculate the output for the entire row
- We will continue this until the values of first channel is computed
- Then for the second channel, we repeat the same process, but now we change the kernel
- Similarly, the values of all the channels are calculated

Explanation of Algorithm 3

Algorithm 3 Algorithm for execution of convolution in the engine

```
1  // Two output rows at a time
2  for r = 1:2:r_o do
3    for c = 1:c_o do
4      // 8 output channels simultaneously
5      for co = 1:8:chl_o do
6        partial_sum[2,8] = 0
7        for c' = 1:c_k do
8          // 8 input channels at a time
9          for ci = 1:8:chl_i do
10
11           // The below part happens in one loop of the core
12           // Hence we replace for with for_unrolled which signifies
13           // that the loop is unrolled over the range of its iterators
14
15         // Temp variable for 384 multiplications
16         // which are accumulated and reduced to 16 partial sums
17         tmp[2,8,8,3] ← 0
18         tmp_reduced[2,8] ← 0
19         for_unrolled co' = co:co+7 do // Use 8 kernels for 8 output channels
20           for_unrolled r' = 1:r_k do        Partial dot product between 1 kernel i.e. [ (1:r_k), c', co' ]
21             for_unrolled ci' = ci:ci+7 do   and 2 inputs i.e. [ (r : r + r_k)     , c + c'-1, ci' ] and
                                                [ (r +1 : r + r_k +1), c + c'-1, ci' ]
22               tmp[1,co',ci',r'] ← input[r+r'-1,c+c'-1,ci']*kernel[co',r',c',ci']
23               tmp[2,co',ci',r'] ← input[r+r',c+c'-1,ci']*kernel[co',r',c',ci']
24             end_unroll
25           end_unroll
26           tmp_reducei[1,co'] ← sum(tmp[1,co',:,:])
27           tmp_reducei[2,co'] ← sum(tmp[2,co',:,:])   For each output channel, add the above
                                                          calculated partial products
28         end_unroll
29
30         sendToAccumulator(tmp_reduce)
31         receiveFromConvolveCore(tmp_reduce)
32         partial_sum ← partial_sum +  tmp_reduce // Element wise sum
33        endfor
34      endfor
          * After line 34, the value partial_sum is sent back to memory
35    endfor
36   endfor
37 endfor
```

Consider an example with input of size (128, 128, 16) and 8 kernels each of size (3, 3, 16). Now we will see how output of (1:2, 1, 1:8) are calculated

1. At first 2 subsets of input are chosen with indices of [1:3, 1, 1:8] and [2:4, 1, 1:8]. The subset of first kernel K1 is chosen with indices [1:3, 1, 1:8]
2. Now, since we are calculating for first output channel, the value of co'(line 19 in above code) is 1
3. Line 20 - 25 is calculating just the product between each pixel value in input and its corresponding kernel value and storing in the array "tmp".
4. Line 26 - 27 is the summation of the products that we calculated in lines 20 - 25.
   So effectively, one loop of lines 20 - 27 gives
   tmp_reduce [1,1] = Input[1:3, 1, 1:8] ***** K1[1:3, 1, 1:8] (co' = 1 and " ***** " indicate dot product)
   tmp_reduce [2,1] = Input[2:4, 1, 1:8] ***** K1[1:3, 1, 1:8]
5. Now once the co' is updated to 2, the process remains the same, but now we use second kernel called K2 and lines 20 - 27 gives
   tmp_reduce [1,2] = Input[1:3, 1, 1:8] ***** K2[1:3, 1, 1:8]
   tmp_reduce [2,2] = Input[2:4, 1, 1:8] ***** K2[1:3, 1, 1:8]
6. Now this process continues until co' reaches 8
7. After the end of loop from 19 - 28, the value of tmp_reduce is sent to accumulator and is added to the variable "partial_sum"
8. After line 33, the above steps from 1 - 7 is repeated, except now the 2 inputs are [1:3, 1, **8:16**] , [2:4, 1, **8:16**] and the kernels will be [1:3, 1, **8:16**]
9. Now we calculate tmp_reduce[1, 1:8] and tmp_reduce[2, 1:8] and sent back to accumulator
10. Once all the 16 channels of the inputs are utilized, now we update the column of the kernels and inputs from 1 to 2 (line 7 in above code)
11. The steps 1 - 9 are repeated and effectively we get the values
    Input[1:3, **2**, 1:8] ***** Kernel_i[1:3, **2**, 1:8],
    Input[2:4, **2**, 1:8] ***** Kernel_i[1:3, **2**, 1:8]
    (where i = 1,2, … 8, as there are 8 kernels) in first iteration of lines 20 - 25 and
    Input[1:3, **2**, 8:16] ***** Kernel_i[1:3, **2**, 8:16],
    Input[2:4, **2**, 8:16] ***** Kernel_i[1:3, **2**, 8:16] in the next iteration
12. These dot product values are sent to accumulator and are added to partial_sum
13. Again we reach line 7 and now calculate the partial dot products with 3rd column of the kernel
14. Now the values will be
    Input[1:3, **3**, 1:8] ***** Kernel_i[1:3, **3**, 1:8],
    Input[2:4, **3**, 1:8] ***** Kernel_i[1:3, **3**, 1:8] in first iteration
    Input[1:3, **3**, 8:16] ***** Kernel_i[1:3, **3**, 8:16],
    Input[2:4, **3**, 8:16] ***** Kernel_i[1:3, **3**, 8:16] in next iteration
15. These values are sent to accumulator and added to the partial_sum

16. After this, the effective value in the partial_sum will be
    Partial_sum[1, i] = Input[1:3, 1:3, 1:16] * Kernel_i[1:3, 1:3, 1:16]
    Partial_sum[2, i] = Input[2:4, 1:3, 1:16] * Kernel_i[1:3, 1:3, 1:16]
    Where, i = 1,2, … 8
17. Partial_sum[1, 1:8] is the value of output [1, 1, 1:8] and
    Partial_sum[2, 1:8] is the value of output [2, 1, 1:8]
18. Now we calculate the values of next 8 output channels and so on