Iowa State University

Department of Electrical and Computer Engineering

Deep Machine Learning: Theory and Practice

EE 526X

Midterm Assignment

Author:
Vishal Deep

Instructor:
Dr. Zhengdao Wang

November 3, 2019

Iowa State University

Paper Title: Memory-Optimal Direct Convolutions for Maximizing Classification Accuracy in Embedded Applications

Introduction and Problem Statement

Machine learning on low-power and low-energy consumption edge devices is growing in popularity. These devices can perform computations locally which helps in increased privacy. But these small edge devices comes with very low memory size range (2KB to 16 KB). Therefore, we have to train the network on a server and can only store weights, biases, activations and configuration parameters on the device. We know that the classification problem gives best results when Convolution Neural Networks (CNN) are used but the CNNs are memory-hungry algorithms. Other works have tried to use alternative algorithms but none of them were able to achieve as much accuracy as CNNs. Therefore, this paper proposes and provides the method to use CNN or very small memory devices of 2KB to achieve best accuracy for the classification problems.

The main idea of this paper is that direct convolutions can be considered as localized operation as pixels in input image are only dependent on some output features. This fact is used in this paper to formulate multiple methods to perform a memory-efficient direct convolutions.

This paper considers two cases which are explained below

- **Same or decreasing channel depth,** which implies that output features are always less than or equal to the input features ($f_{out} \leq f_{in}$). This case is much easier to handle because the generated output pixels can be replaced with the staled input pixels which are already used for convolutions and hence can be removed.
- **Increasing channel depth** means that the generated output features are more than the input features ($f_{out} > f_{in}$). This case is more challenging as there will not be enough space in the memory to accommodate all the output pixels.

  These two cases are depicted in the fig. 1.

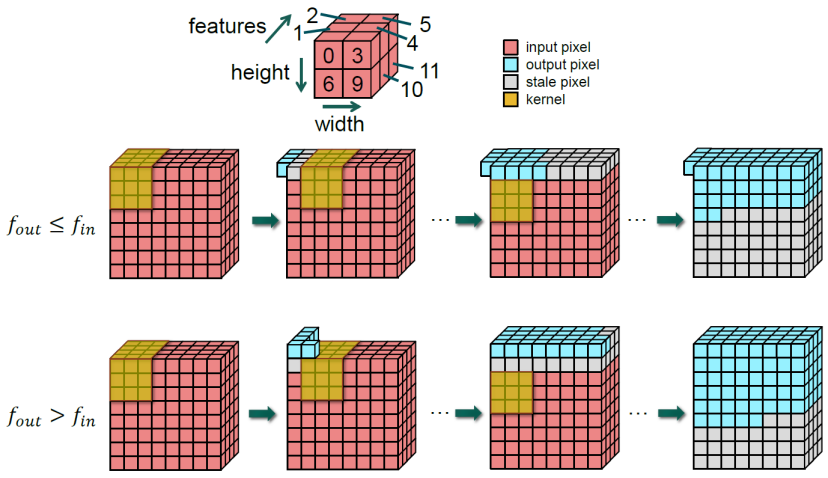

Figure 1: Arrangement of pixels in memory (top) and considerations for where to place output activations based on how fin compares to fout. [1]

Methodology/Approach

The authors of this paper proposed three approaches to perform direct convolution keeping in mind memory efficiency.

1. **Replace Strategy:**
   In the replace strategy, the computation of output pixels are performed in row-wise fashion. This approach is able to produce k-1 stale input pixels, where k in kernel size, when an output pixel in bottom row is computed but this available space can't be used for storing output pixels because of the inconvenient location in the memory. Therefore this approach is not really memory efficient. Therefore, another strategy called "herringbone" is proposed to provide memory optimal solution.

2. **Herringbone Strategy:**
   This approach uses the fact that if we compute the edge pixels of the input image we can free out more memory than other schemes. The herringbone strategy computes the convolution in row-column-row-column fashion as shown in fig. 2. The memory optimal solution of the herringbone strategy must have an efficient way to access these stale/free memory locations. This paper also proposed two approaches to solve this problem:

   - One solution is to shift the input pixels in the memory every time a input pixel becomes stale. This will fill the stale spot in the memory but it is computationally more expensive.
   - The other solution is to do a transpose whenever a switching takes place between row and column. This approach provides less computational complexity because transpose operation has to be done on entire rows and columns as compared to shift method.

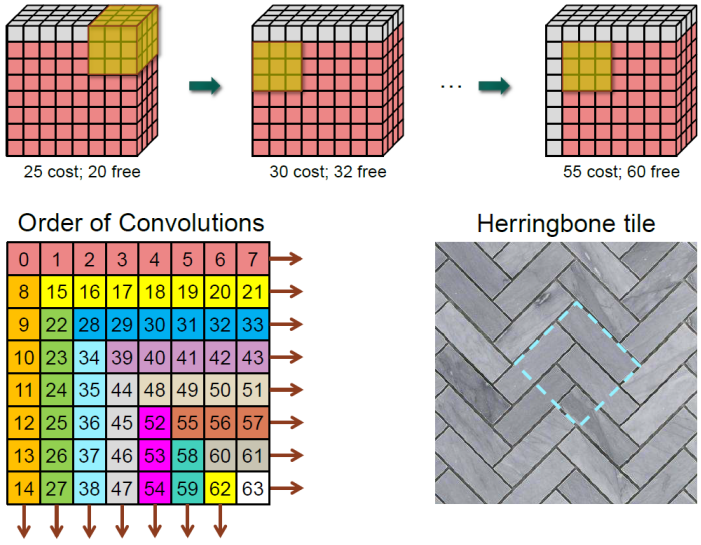

Figure 2: Herringbone Strategy [1]

3. **Single transpose Strategy:**
   The problem with herringbone method is that it's computationally expensive because of the operations like shifting and transpose. In the single transpose strategy, first it processes several rows and then take a transpose of columns and make them rows. After that remaining rows which were columns before are processed. The paper has shown that the memory requirements for herringbone and single transpose methods are approximately similar but computational cost is significantly reduced in the later case.

Implementation

The methods or strategies described above are still computationally expensive therefore author also proposed memory-efficient operations used in the above mentioned techniques.

1. **Memory-efficient Transpose**:
   In this method, the transpose is not done in one cycle rather in multiple disjoint cycles. It uses a successor function to map each location with other locations. The transpose can be obtained by recursive application of this function in multiple cycles. The example of this transpose is given in the fig. 3.

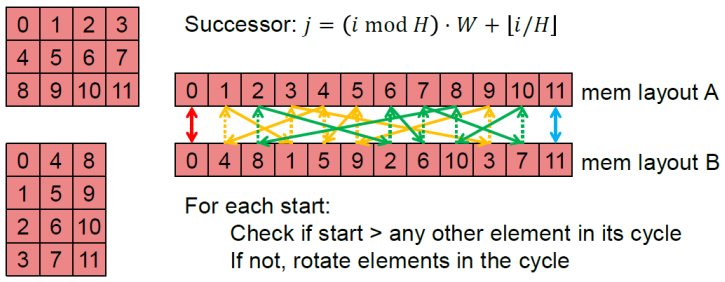

Figure 3: Transpose example [1]

2. **Memory-efficient Inverse Herringbone**:
   This is a similar kind of approach but uses a different successor function. This operation is only valid when there is a square matrix. When there is need to move an element from un-permuted index to a permuted index then indices could be split into upper and lower triangular matrices which helps in calculating the function for this approach.

Results

This paper evaluated the proposed strategies on the an Arduino which use a ATmega328P chip. The input images and parameters are fed serially to the board and output is also taken from the serial port as shown in the fig. 4. The network is trained on tensorflow/keras with 50 epochs in floating point and 200 epochs in 4-bit quantized training. The estimator used is straight-through

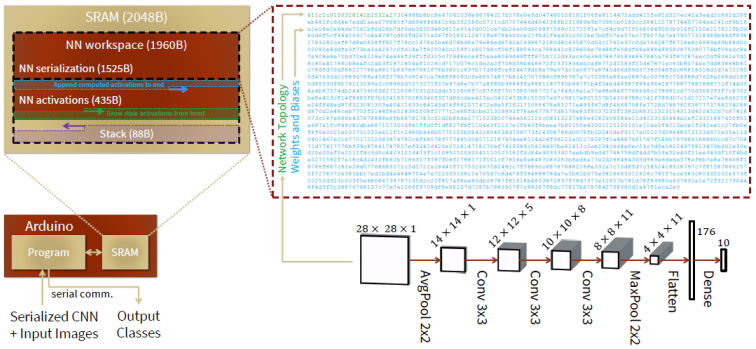

Figure 4: Implementation of CNN on Arduino [1]

and ALT-training. The verification of results is also done for intermediate activations for all 10000 test images. The results are summarized in the table 1.

RESULT	VALUE
MEMORY	434.5 B (ACT)/ 1512.5 B (WTS)
TOTAL SERIALIZED	1960 B
PROGRAM SIZE	6514 B
INFERENCE TIME	684 MS (NOT OPTIMIZED)
ACCURACY	99.11 % (DEV) / 99.15 % (TEST)

Table 1: Results Summary

The fig. 5 shows the pareto curves for randomly selected different architectures with different parameters. The models are trained for 5 epochs for floating point weights/activations (red dots) and 5 epochs with quantized weights/activations (blue dots). From fig. 5, we can see that as the memory size for weights and activation increases the validation error decreases.

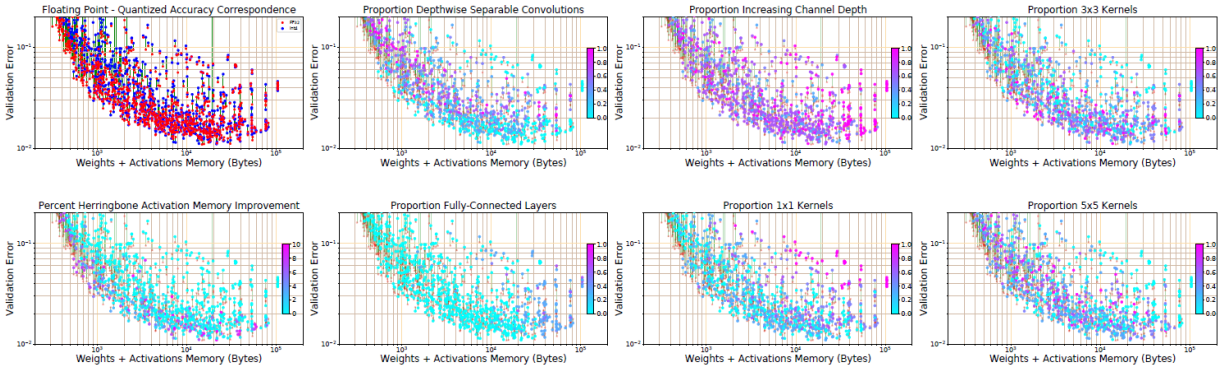

Figure 5: Pareto curves for randomly selected architectures with different architectural features. [1]

Paper Critique

- The paper only provides the analyses of memory consumption of weights and activations but it doesn't provide the detail of additional memory consumed by other things such as loop variables etc.
- The paper only evaluates the MNIST data-set. It would have been nicer if the analyses of more data-set was done. This would have strengthens the results obtained in this paper.
- As described in the paper, the herringbone and single transpose methods can only be useful for 2D and 3D convolutions where ($f_{out} > f_{in}$), k > 1 and kernel size (k) is not too small as compared to feature map dimensions.
- According to the pareto curves in fig. 5, there is a sweet spot of memory consumption around 2 KB that means if they consumed a little bit more or less memory then the accuracy of the system would have been less. And this impact is very large on both sides of the 2 KB memory spot.

Future Directions

- In the future, I would like to see the implementation of these strategies on different boards not just Arduino. There are multiple boards which have limited power and memory which are good for applications in these categories e.g. TI's MSP430 [2], ARM M0 boards [3] etc. Not only different boards, I think different data-sets can be used to evaluate the techniques strongly.
- I would also like to see the comparison of proposed techniques with the classification algorithms of the tensorflow lite [4] on the same device. The tf lite is made for small low-power and low-memory devices. The comparative analyses would include the computation as well as memory efficiency in the small edge devices.
- Another approach I can think of is to use distributed computing where we can distribute/share the computational load as well as the memory load between the same or different types of edge devices. This will help in computation of much larger data-sets and in decent time.

References

[1] A. Gural and B. Murmann, "Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications," in *International Conference on Machine Learning*, pp. 2515–2524, 2019.

[2] T. Instruments, "Msp430g2 launchpad development kit," 2014.

[3] A. Developer, "Cortex-m0," 2019.

[4] T. Flow, "Deploy machine learning models on mobile and iot devices," 2019.