CS 758: Advanced Topics in Computer Architecture

Lecture #14: Reuse

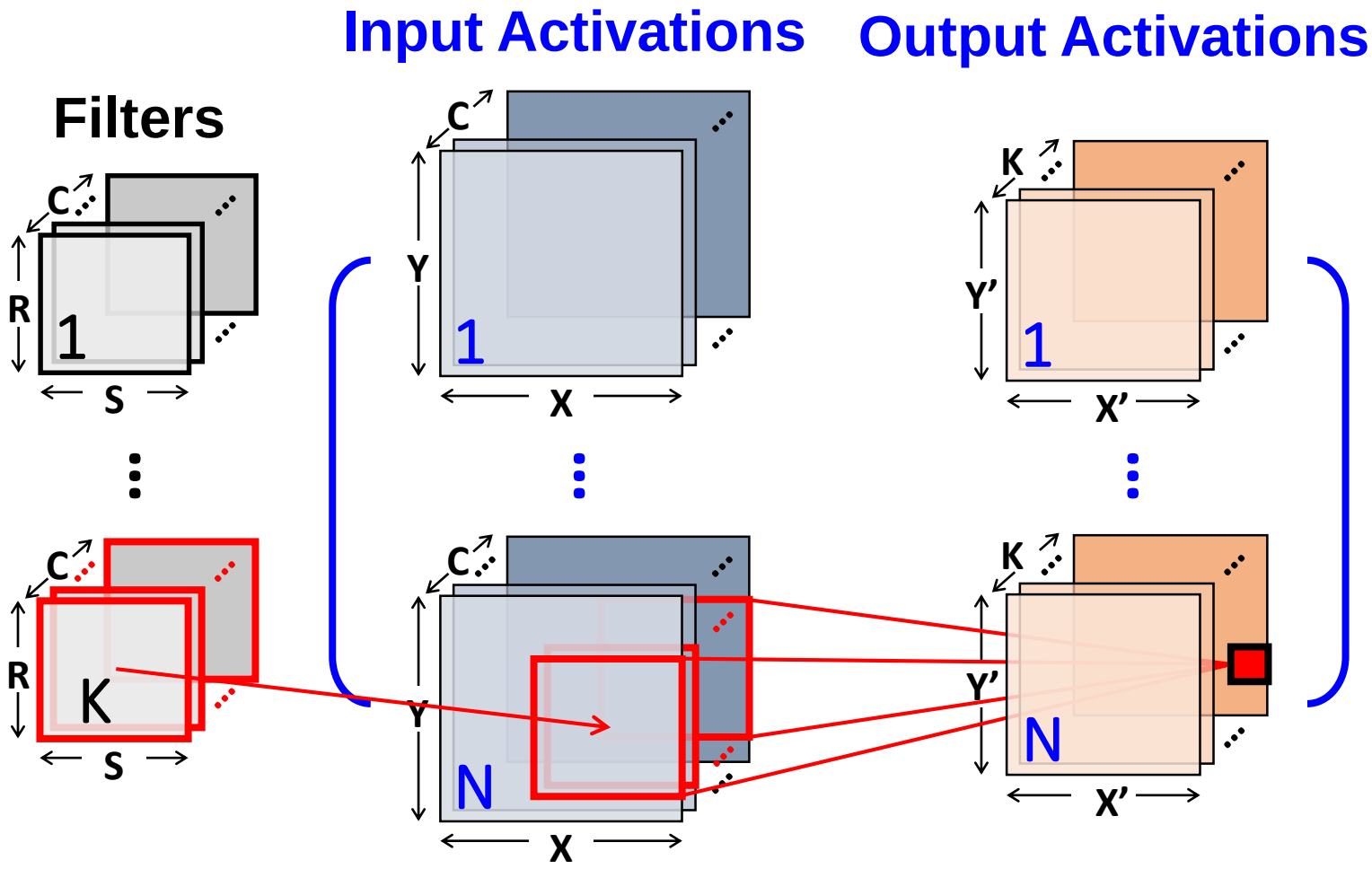
Professor Matthew D. Sinclair

Some of these slides were developed by Tushar Krishna at Georgia Tech
Slides enhanced by Matt Sinclair

Announcements

- Project Progress Report due next Tuesday at 9 AM

CNN Computation



CNN Loop-nest

```
for(k=0; k<K; k++) { // Weight Filters
  for(c=0; c<C; c++) { // IFMap/Weight Channels
    for(y'=0; y'<Y'; y'++) {  // Output feature map row
      for(x'=0; x'<X'; x'++) {  // Output feature map column
        for(r=0; j<R; j++) {  // Weight filter row
          for(s=0; i<S; i++) {  // Weight filter column
            O[k][y'][x'] += W[k][c][r][s] * I[c][y'+r][x'+s]}}}}}}}
```

Output-Centric

```
for(k=0; k<K; k++) { // Weight Filters
  for(c=0; c<C; c++) { // IFMap/Weight Channels
    for(y=0; y<Y; y++) {  // Input feature map row
      for(x=0; x<X; x++) {  // Input feature map column
        for(r=0; j<R; j++) {  // Weight filter row
          for(s=0; i<S; i++) {  // Weight filter column
            O[k][y-r][x-s] += W[k][c][r][s] * I[c][y][x]}}}}}}}
```

Input-Centic

Data Reuse opportunities

- Dataflows
  - Mapping
    - Weight: aka weight stationary
    - Input: aka input stationary
    - Partial Sums: aka output stationary
  - Loop Scheduling and Tiling
    - Maximize on-chip buffers and reduce off-chip data transfers

- More Reuse Opportunities
  - FusedCNN (MICRO 2016): Reuse outputs across layers
  - UCNN (ISCA 2018): Reuse unique weights across filters
  - DianNao (ASPLOS 2014): accumulate partial sums in registers (output stationary)

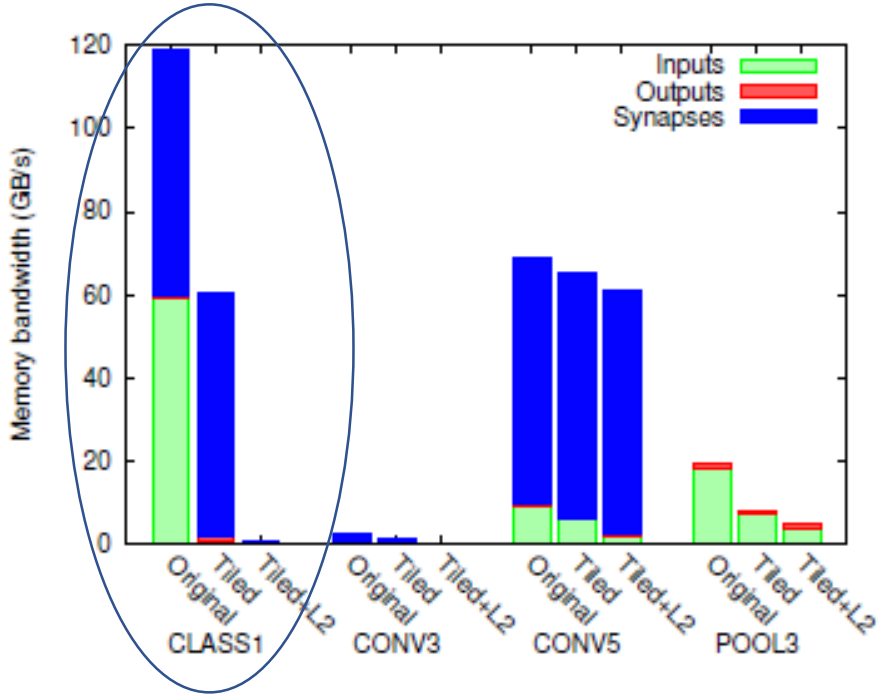
DianNao Summary (Best Paper, ASPLOS ‘14)

- Machine learning is becoming pervasive
  - **A couple of ML algorithms cover many applications**
  - Heterogeneous systems should include ML accelerators
- Previous work ignores memory constraints
  - Memory **must** be a first-order design constraint
- Heavily optimized ML accelerator (DianNao)
  - 3.02 mm^2 at 65 nm, 485 mW footprint
  - 452 GOP/s
  - **117.9X** faster, **21.1X** more energy efficient

Attempt #1: Software NN

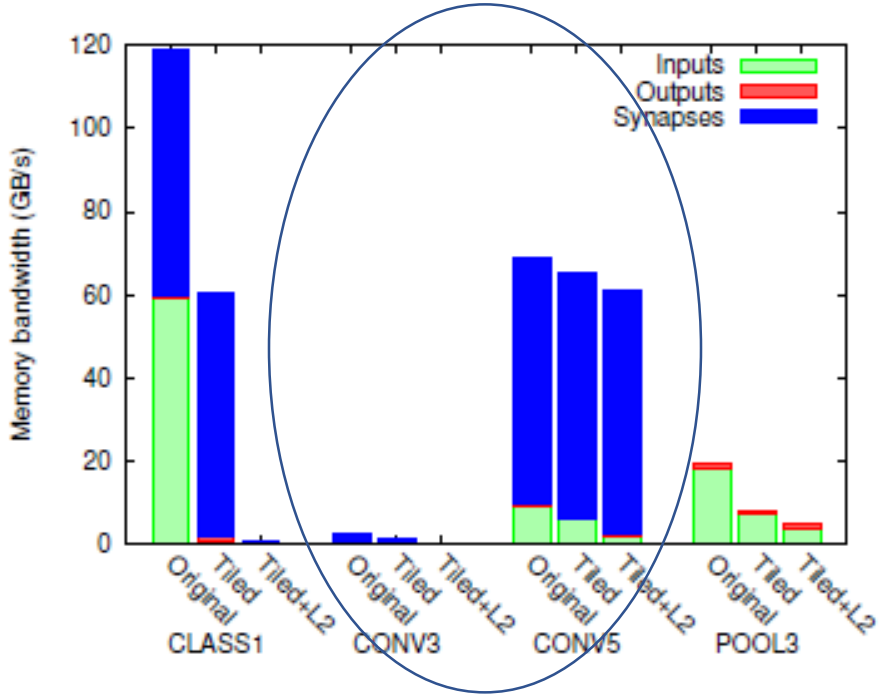
- Used to show the high amounts of memory traffic
  - Motivates hardware design choices
- Classifier Layer:
  - Each output has 10K – 100K of input neurons!
  - Therefore cannot fit everything in the L1 cache
  - Solution: tile loops to get better locality
    - Synapses

Software NN: Classifier Layer



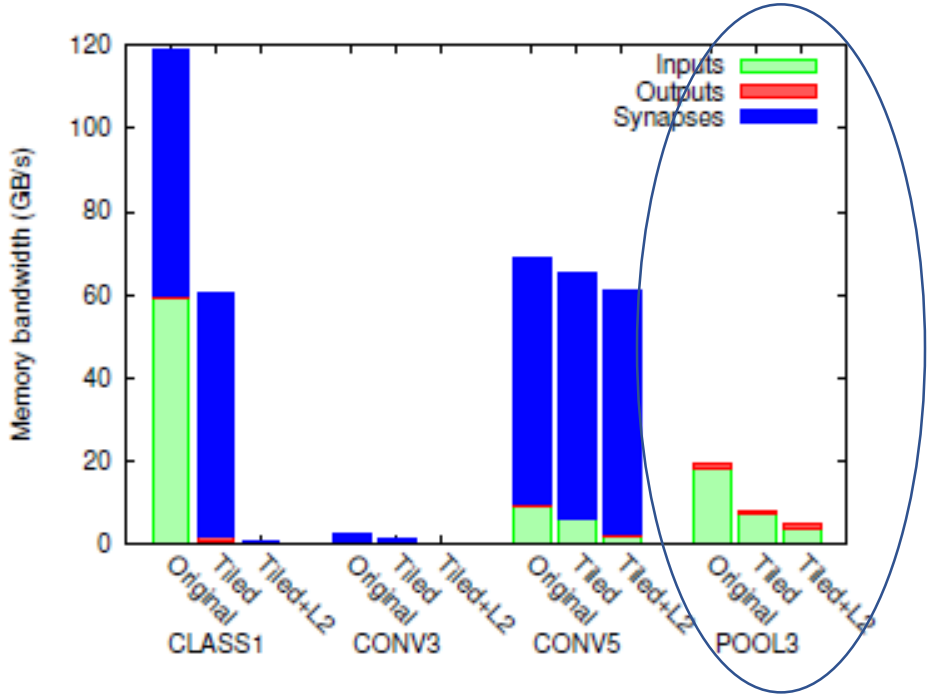
- Each output has 10K – 100K of input neurons!
- Therefore cannot fit everything in the L1 cache (*Original*)
  - Solution: tile to get better locality and reduce mem B/W (*Tiled*)
  - Synapses: reuse across invocations with larger L2 (*Tiled+L2*)

Software NN: Convolutional Layer



- Uses a sliding window to scan input layer (*Original*)
- Tiling helps here too but not as much (*Tiled*)
- Synapse weights can again be reused across iterations
  - Solution: larger L2 (*Tiled + L2*) – but often synapses too big for L2

Software NN: Pooling Layer



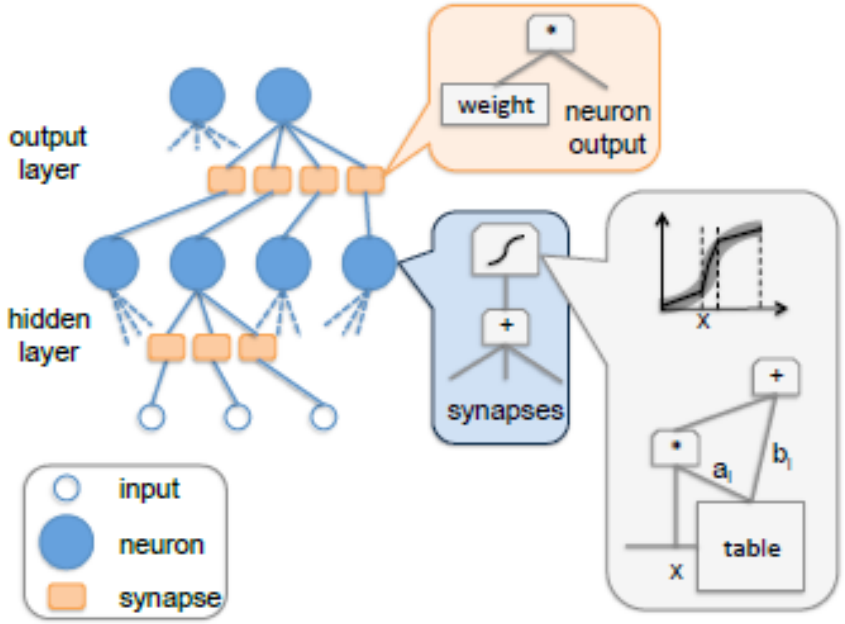
- No synapse weights to store, so less opportunities for reuse
- Tiling still increases reuse for inputs (*Tiled, Tiled+L2*)
  - Larger L2 cache helps because lots of input neurons

Software NN: Takeaways

- Lots of intra- and inter-layer memory traffic!
  - Sometimes memory requirements exceed L1 or L2 size
  - Solutions:
    1. Tile loops to increase reuse of input neurons
    2. Increase L2 cache size to create reuse of synapse weights

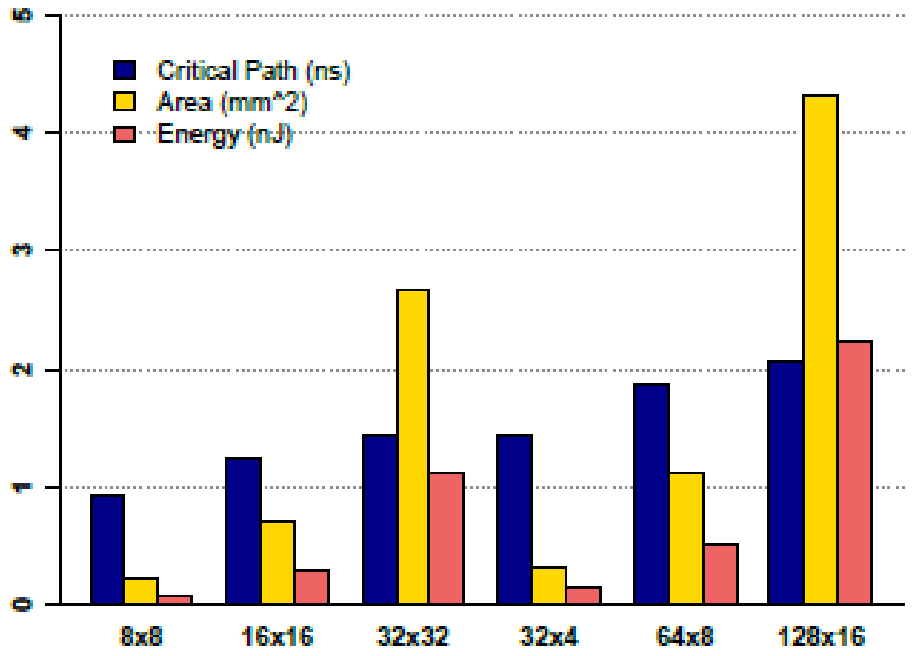
- Need better solution that focuses on memory requirements
- Can we do better with hardware NNs?

Attempt #2: Naïve Hardware NN



- Naïve (simple) hardware approach:
  - Lay out all neurons and synapses for all layers
    - Neurons: logic circuits
    - Synapses: latches or RAMs
  - Matches conceptual NN design
  - Memory only used for input and storing results

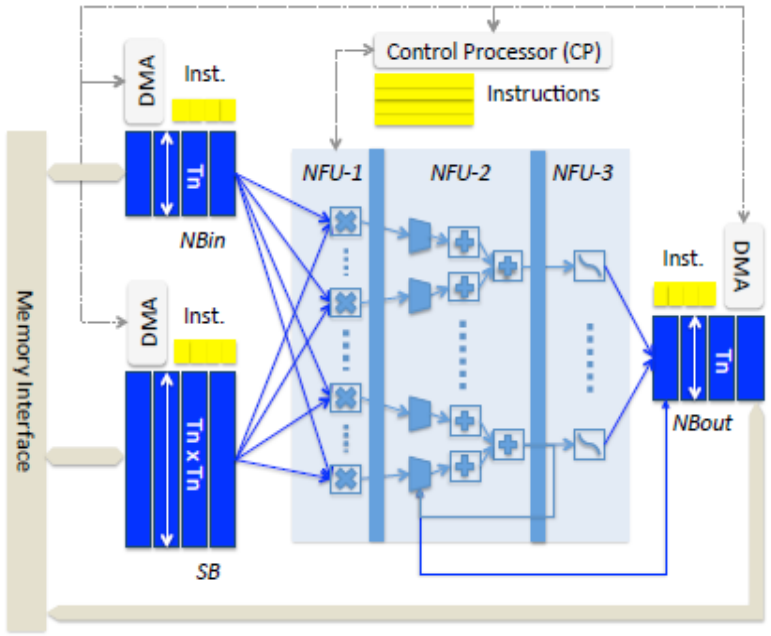
Naïve Hardware NN: Results



- Advantages:
  - Simple design
  - Short distance between layers (decreased energy)
  - Fast!
- Disadvantages:
  - Huge area requirements
  - Limited number of neurons and synapses can be used

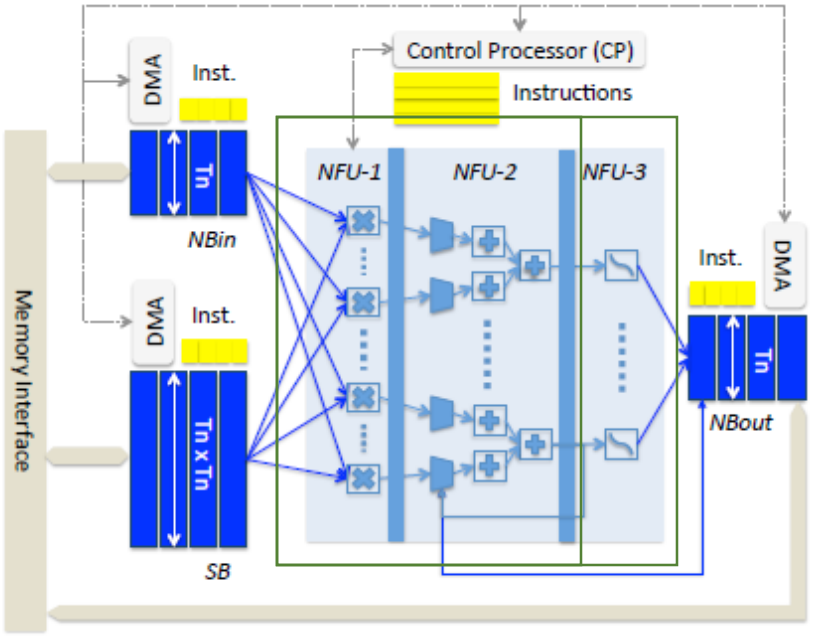
Improves memory requirements but ignores area!

Attempt #3: Large Hardware NN



- Main components:
  - Computational block for all calculations (*NFU-**)
  - Input buffer for input neurons (*NBin*)
  - Output buffer for output neurons (*NBout*)
  - Buffer for synapse weights (*SB*)

Large Hardware NN: Pipeline

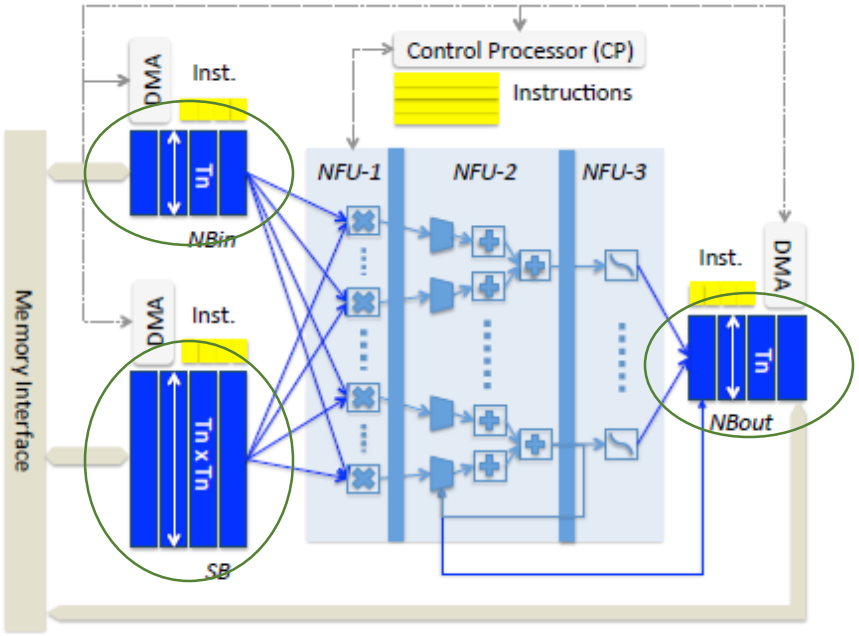


- **Decompose layers to reduce critical path delay**
- Classifier (FC): multiply weights and inputs, sum, sigmoid
- Convolutional: multiply weights and inputs, sum, sigmoid
- Pooling: Combine using average or max

Large Hardware NN: Arithmetic

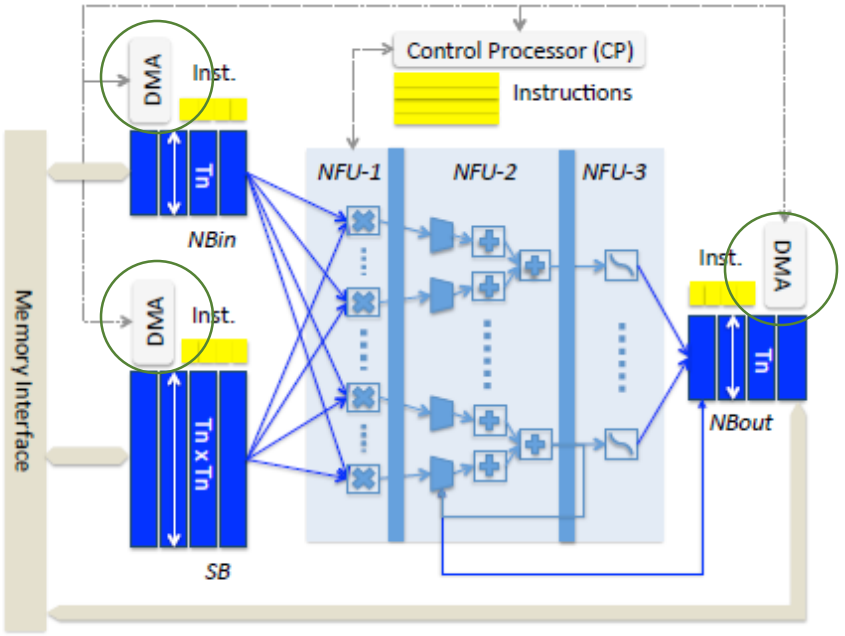
- NN pipeline heavily uses arithmetic
  - Need to make these operations efficient!
- Optimizations:
  - Adds, Multiplies:
    - 16-bit operations instead of 32-bit operations
    - Small quality degradation (NNs are robust, fault tolerant)
  - Sigmoid (NFU-3): use linear interpolation
    - Reduces to multiplies, additions, and SRAM lookups

Large Hardware NN: Storage



- **Split buffers** to use appropriate width, size for each
- Use **scratchpads** to store appropriate data
  - Scratchpads are more energy efficient than caches
  - Data guaranteed to be in scratchpad (more reuse)
  - Have to manually re-write code to use scratchpads

Large Hardware NN: Locality



- Add **DMAs** to prefetch scratchpad data, exploit locality
  - DMAs prefetch into separate FIFO for each scratchpad buffer
- Make NBin a **circular buffer** to reuse data across tiled chunks
- **Dedicated registers** to store partial sums

Conclusion

- Heterogeneous systems are becoming common
  - Which accelerators they should include is an open question
  - This paper: heterogeneous systems should include ML accelerators!
- **A couple of ML algorithms cover many applications**
  - Memory **must** be a first-order design constraint
- Created a heavily optimized ML accelerator (DianNao)
  - Somewhat like an ASIC
  - Significantly improves performance and energy efficiency

Key Takeaways

Advantages

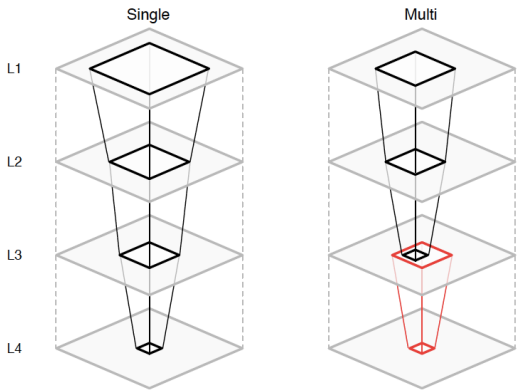
- Focuses on memory
- Step-by-step design walkthrough
  - Explain what the key facets are
  - Synthesized design, real #'s
- Exploits fact that CNNs and DNNs work for many applications
  - Allows a single HW design

Disadvantages

- How reusable is their design?
- Not very programmable
  - NPU work from Washington is much better in this respect
- Requires significant baseline understanding of concepts

Fused-CNN

- What is the basic idea?
- Trade-offs?
  - What about weight or input reuse?
  - Reuse vs Recompute
- Design Choices



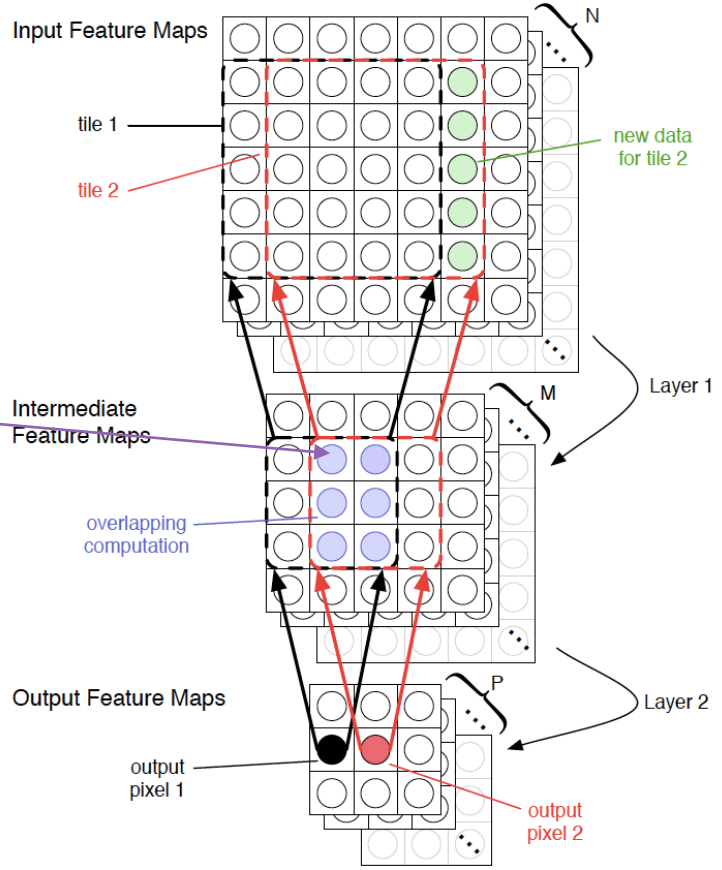
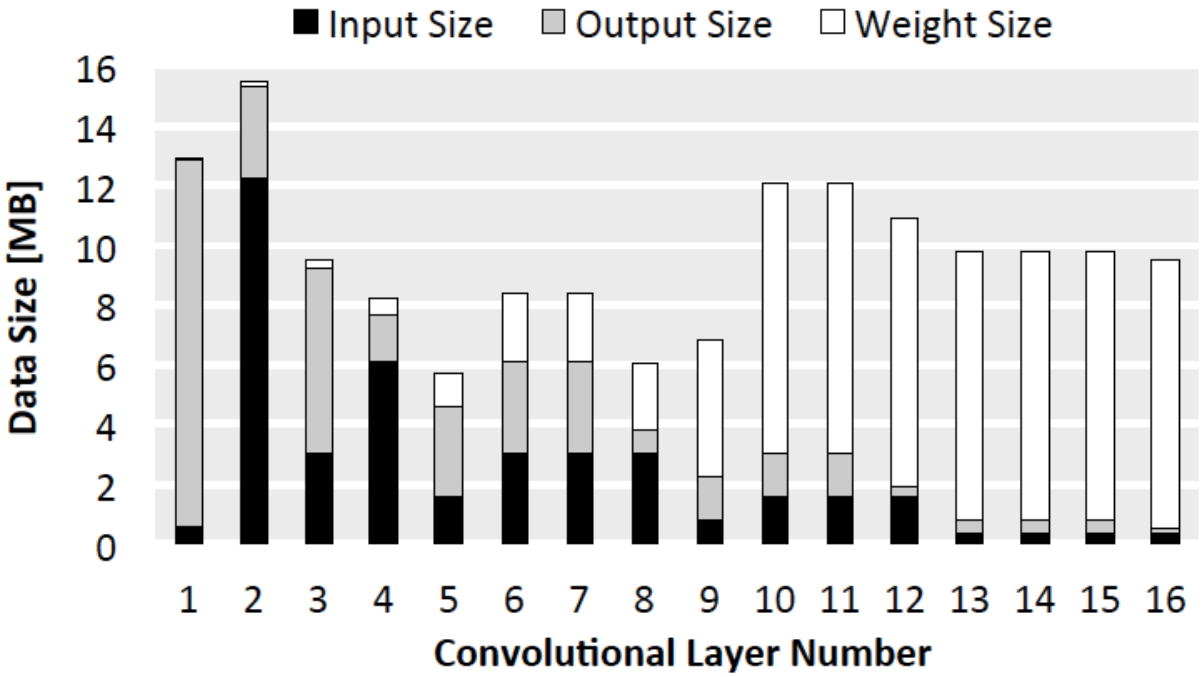


Fig. 3. Example of fusing two convolutional layers.

When does it make sense?

- Large Output Sizes
  - Early layers of CNNs



- Few inputs/weights and generating entire output pixel
  - E.g., convolutions

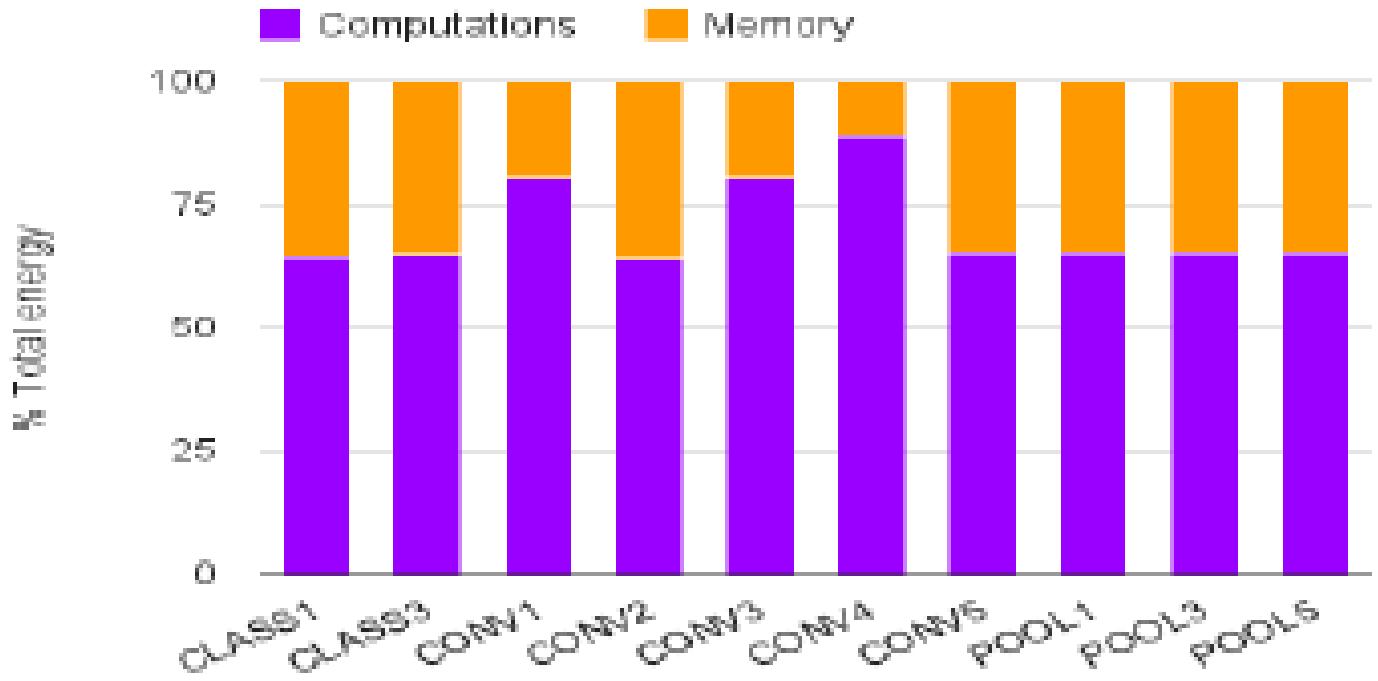
BACKUP

Naïve Hardware NN: Takeaways

- Advantages:
  - Simple design
  - Short distance between layers (decreased energy)
  - Fast!

- Disadvantages:
  - Huge area requirements
  - Limited number of neurons and synapses can be used

Results: SIMD Breakdown



Background: Layers