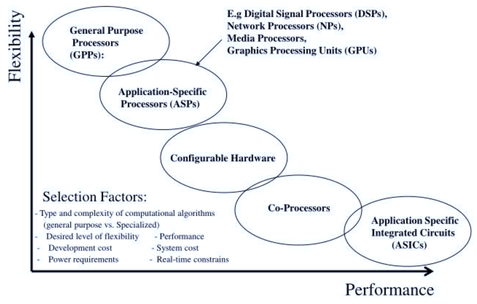
# Designing a RISC-V Convolutional Neural Network Co-Processor

Jacob Bruhn, Hope Hong, Alison Kennedy, Oguzhan Yilmaz
Advisor: Dr. Alenka Zajic | Mentor: Nader Sehatbakhsh

## Introduction

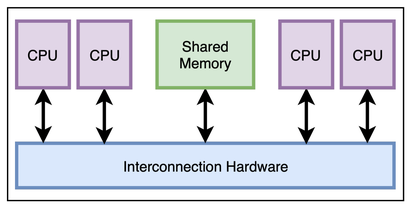
- Convolutional Neural Networks (CNN) are often used in machine learning due to their success in classifying images.
- However, CNN computation is intensive for the main processor.
- Computer architects develop accelerators to relieve the main processor of CNN computation.

**Computing Engine Choices**

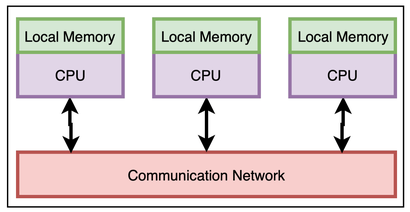



## Background

**Accelerator Design**




**Tightly Coupled**
+ Connection to L1 cache
+ No invocation overhead
- Dependent on CPU



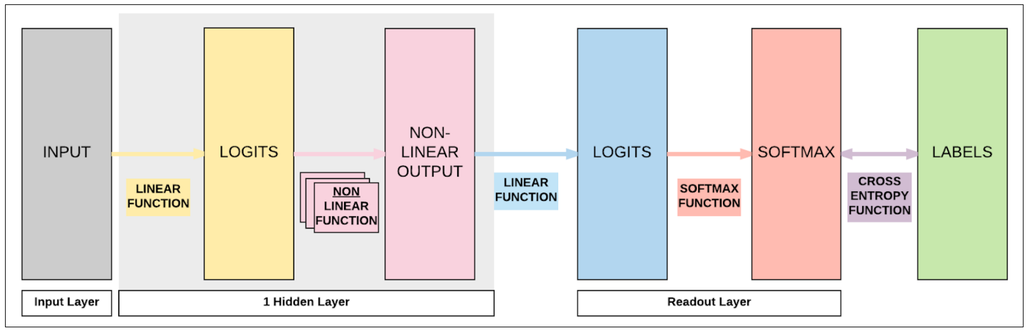

**Loosely Coupled**
+ CPU independent
- Slow memory access
- Little process interaction

**RISC-V is an open instruction set architecture allowing anyone to create processor architectures**

## Design in Chisel

+ Open-source high level language and object oriented programming
+ Supports advanced hardware design
+ Implements RISC-V ISA

**Accelerator Design**




*CNN comprises of four different types of layers: Convolutional, Rectified Linear Unit, Pooling and Fully Connected Layers*

```
object main {

   def main(args:Array[String]) {

   val input = Source.fromFile("vector.txt").getLines.toVector;
   val weights = Source.fromFile("weights.txt").getLines.toVector;
   val lines = Source.fromFile("layers.txt").getLines.toVector;
   val out = Output(Vec(Seq.fill(len) {UInt()}));

   for (label <- lines)
{
   if(label == "Relu") {
       out = Relu(input, weights);}
   else if(label == "MaxPool") {
       out = MaxPool(input, weights);}
   else if(label == "FullyConnected")  {
       out = FullyConnected(input, weights);}
   else if(label == "Convolutional")  {
       out = Convolutional(input, weights);}
   else    {
       println("Not a valid layer."); }
   in = out;
}
}
}
```

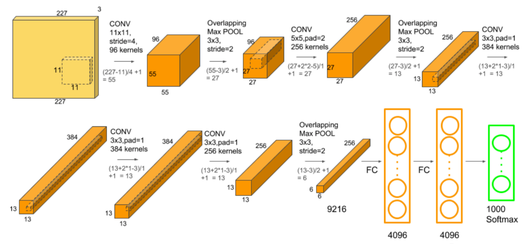
**Wrapper**
- Accelerator interface
- Receives processor data
- Connects all layers

**Layer Implementations**
- Code synthesized to hardware
- Programs circuit modules and wiring perform layer operations

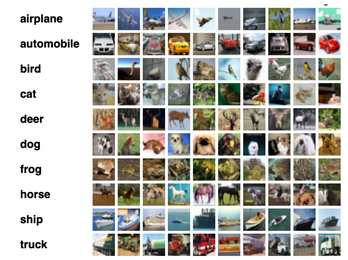
```
import chisel3._
import chisel3.util._
class FullyConnected(val n:Int) extends Module{
        val io = IO(new Bundle {
        val in1 = Input(Vec(n, UInt(4.W)))
        val in2 = Input(Vec(n, UInt(4.W)))
                val out = Output(UInt(8.W))
        })
   val As   = Array.fill(n-1)(Module(new Adder(8)).io)
   val mulvals = Reg(Vec(n, UInt()))
   val mul1 = Array.fill(n)(Module(new Mul()))
   for (i <- 0 until n) {
       mul1(i).io.x := io.in1(i)
       mul1(i).io.y := io.in2(i)
       mulvals(i) := mul1(i).io.z
   }
   As(0).A := mulvals(0)
   As(0).B := mulvals(1)
   for (j <- 0 until n-2) {
       As(j+1).A := As(j).Sum
       As(j+1).B := mulvals(j+2)
   }
   io.out := As(n-2).Sum
}
```

## Results

***Network***:
Alex-Net
(5 conv,
3 FC)




***Input Dataset***:
CIFAR-10




***Implementation:*** +80% accuracy for *top-5 category*

| Bit-width | Device | Freq (MHZ) | Through (GOPs) | LUT (K) | Memory (MB) |
|---|---|---|---|---|---|
| Fixed-16 | Virtex-7 | 120 | 160 | 138 | 18.2 |

## Future Directions

- Extend the CNN co-processor as a secure CNN accelerator to classify private images despite training on an untrusted server.
- Study the effect of side-channel leakage (e.g. variation in power, electromagnetic emanations, etc.) on the security and privacy of a RISC-V based CNN co-processor.
- Exploring *reverse-engineering* methods on the co-processor to infer the CNN model and/or input data.