ECE382M.20: System-on-Chip (SoC) Design

Lecture 1 – Project Overview

Andreas Gerstlauer
Electrical and Computer Engineering
The University of Texas at Austin
gerstl@ece.utexas.edu

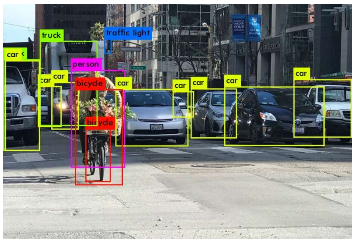
The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 1: Outline

- **Marketing requirements**
  - Market focus, product description
  - Cost metrics, product features
- **Product requirements**
  - Deep learning
  - Hardware acceleration
- **Project description**
  - Deep/Convolutional Neural Networks (DNNs/CNNs)
  - Object recognition
  - You Only Look Once (YOLO) CNN
  - Hardware and software development tasks

Market Focus

- **Visual object recognition**
  - Computer vision for drones, self-driving cars, home automation, …
  - ➢ Camera-based automotive driver assistance systems (ADAS)

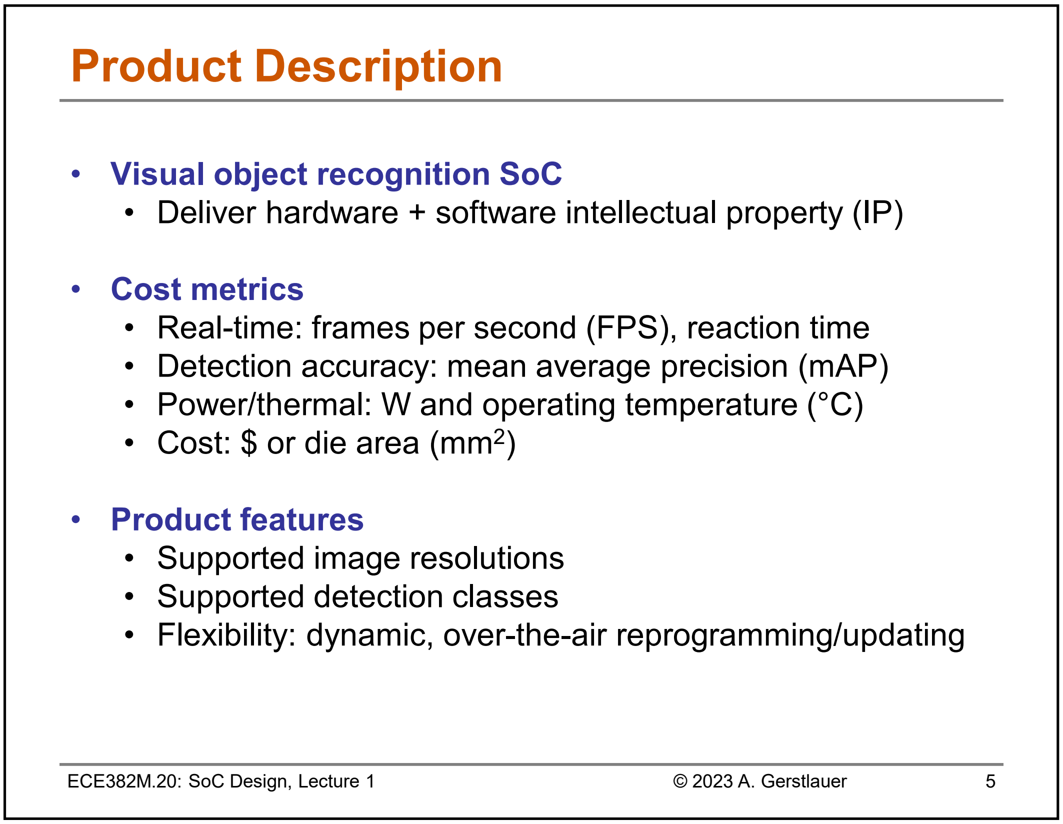


Source: Jonathan Hui

➢ **What problem are we trying to solve?**
  - Standard camera for
    - Collision avoidance
    - Lane tracking/keeping
    - Traffic sign recognition
    - …
  - Detect, locate and classify objects in video stream
    - Bounding boxes
    - Types of objects

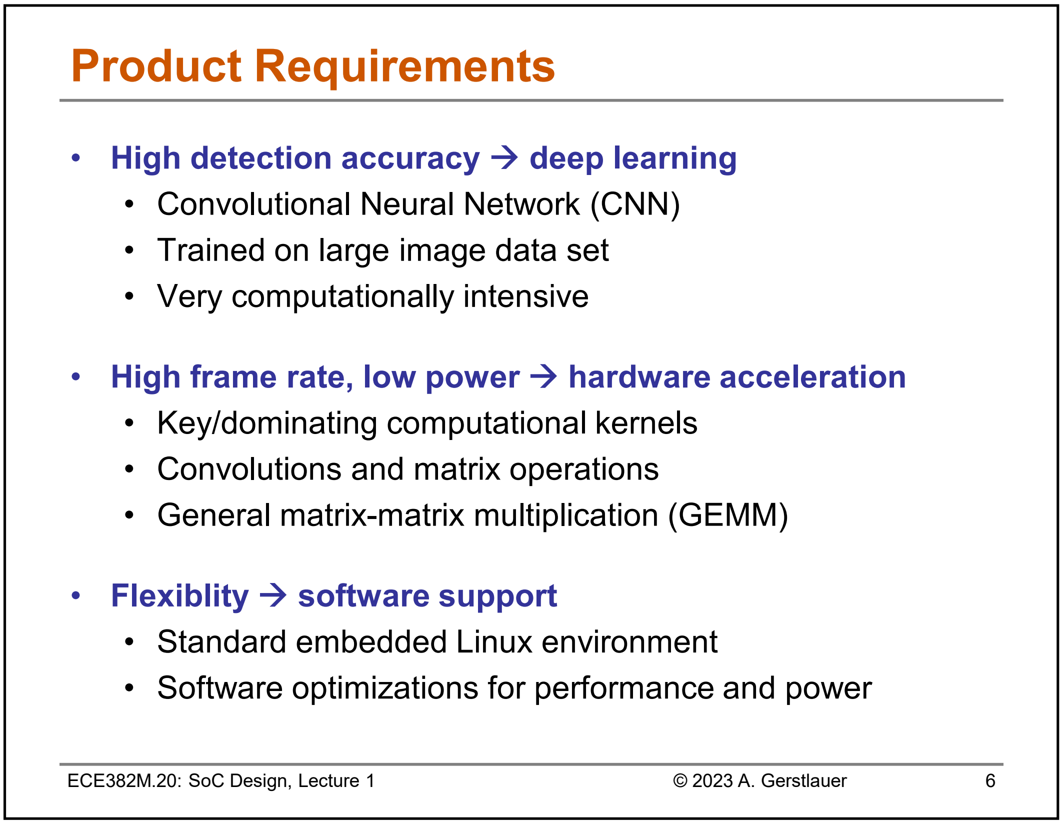
Competition

- **MobilEye (an Intel company)**
  - http://mobileye.com
  - Custom ASIC/SoC solution
- **Movidius (an Intel company)**
  - https://www.movidius.com/
  - Custom ASIC/SoC solution
- **NVIDIA DRIVE**
  - https://www.nvidia.com/en-us/self-driving-cars
  - ARM+GPU based solution
  - Used by Tesla



Product Description

- **Visual object recognition SoC**
  - Deliver hardware + software intellectual property (IP)
- **Cost metrics**
  - Real-time: frames per second (FPS), reaction time
  - Detection accuracy: mean average precision (mAP)
  - Power/thermal: W and operating temperature (°C)
  - Cost: $ or die area (mm^2)
- **Product features**
  - Supported image resolutions
  - Supported detection classes
  - Flexibility: dynamic, over-the-air reprogramming/updating



Product Requirements

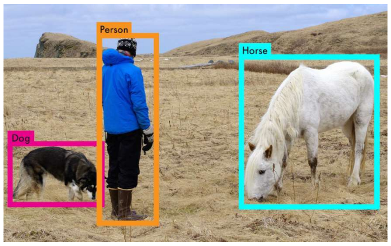
- **High detection accuracy → deep learning**
  - Convolutional Neural Network (CNN)
  - Trained on large image data set
  - Very computationally intensive
- **High frame rate, low power → hardware acceleration**
  - Key/dominating computational kernels
  - Convolutions and matrix operations
  - General matrix-matrix multiplication (GEMM)
- **Flexiblity → software support**
  - Standard embedded Linux environment
  - Software optimizations for performance and power

Objection Detection using Deep Learning

- **Classification vs. detection**

Image classification, global feature

Object detection, classification + localization

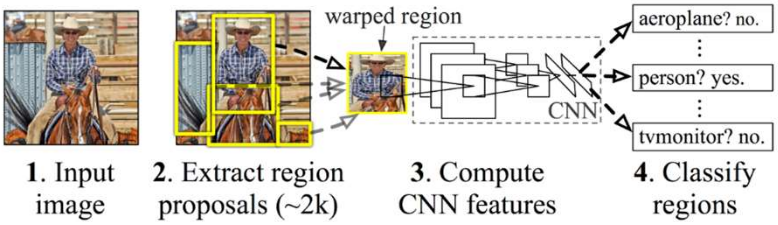


- Convolutional neural networks (CNNs) widely used for image classification
- Sliding windows of different size/shape + CNN-based classification for brute-force, naïve object detection

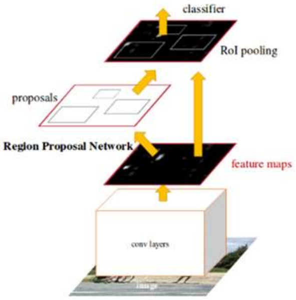
Object Recognition (1)

- **Region based (Fast/Faster/Mask R-CNN)**

R-CNN: ***Regions with CNN features***

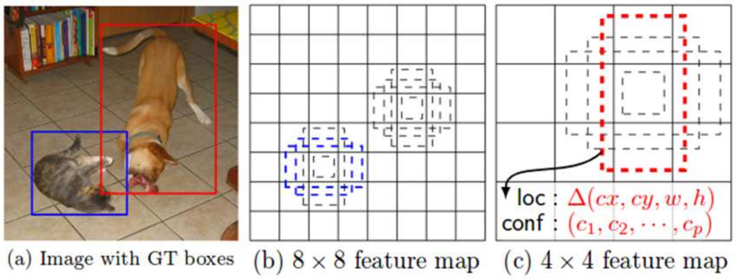


- Fast versions by sharing convolutional layers
- Common feature extraction for region proposal and classification



Object Recognition (2)

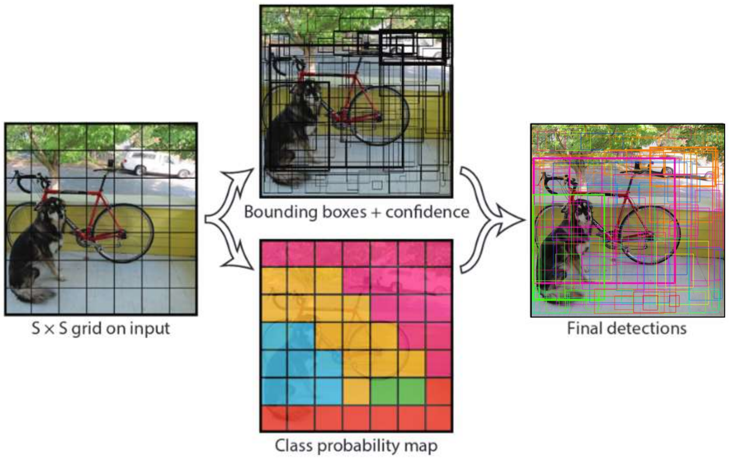
- **Single Shot Multibox Detector (SSD)**


loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

- Apply windows of fixed size and shape at multiple scales
- Detect both bounding box and class within window
- Predict likelihood of different box/class combinations

Object Detection (3)

- **You Only Look Once (YOLO)**
  - Single grid/scale, predict arbitrary bounding box (and class)



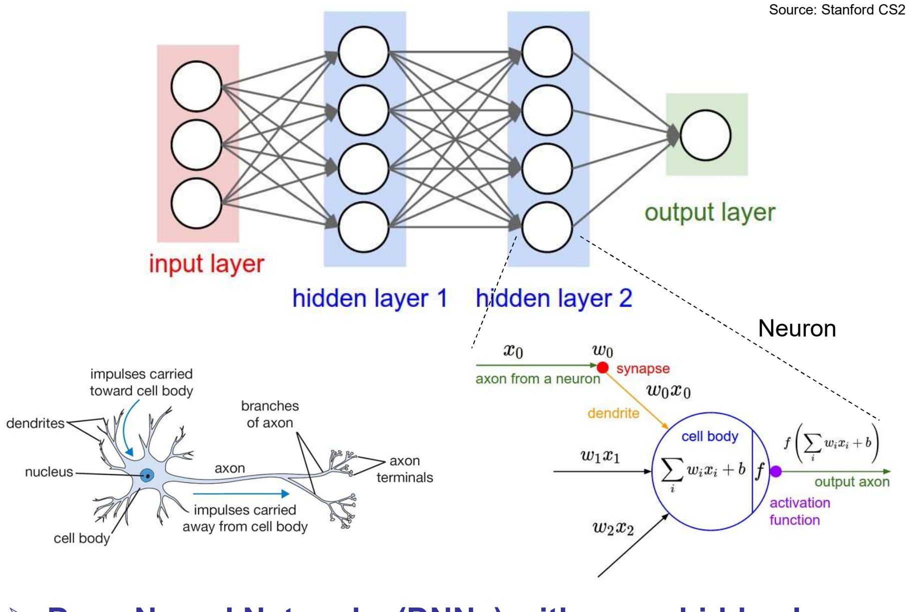
You Only Look Once (YOLO)

https://pjreddie.com/darknet/yolo/

- **Default implementation on top of Darknet**
  - General open-source CNN framework/library in C
- **Also available for other deep learning frameworks**
  - PyTorch, Caffee2 [Facebook], TensorFlow [Google]

Traditional Neural Networks

Source: Stanford CS231n

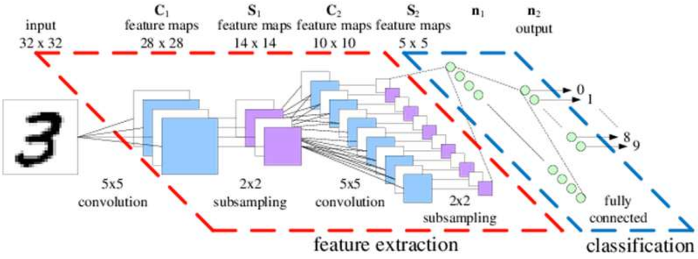


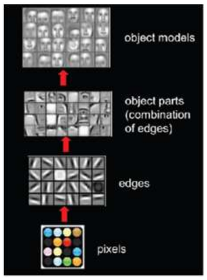
➢ **Deep Neural Networks (DNNs) with many hidden layers**

Convolutional Neural Networks (CNNs)

- **Different types of layers**
  - Convolutional: convolutions with trainable filters
  - Rectified linear units (ReLU): elementwise function
  - Pooling: non-linear down-sampling
  - Fully connected: traditional neural networks

} usually combined

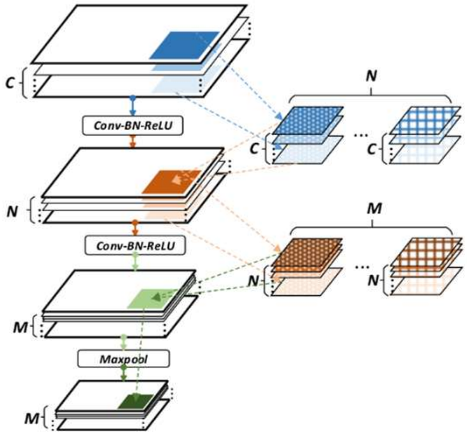


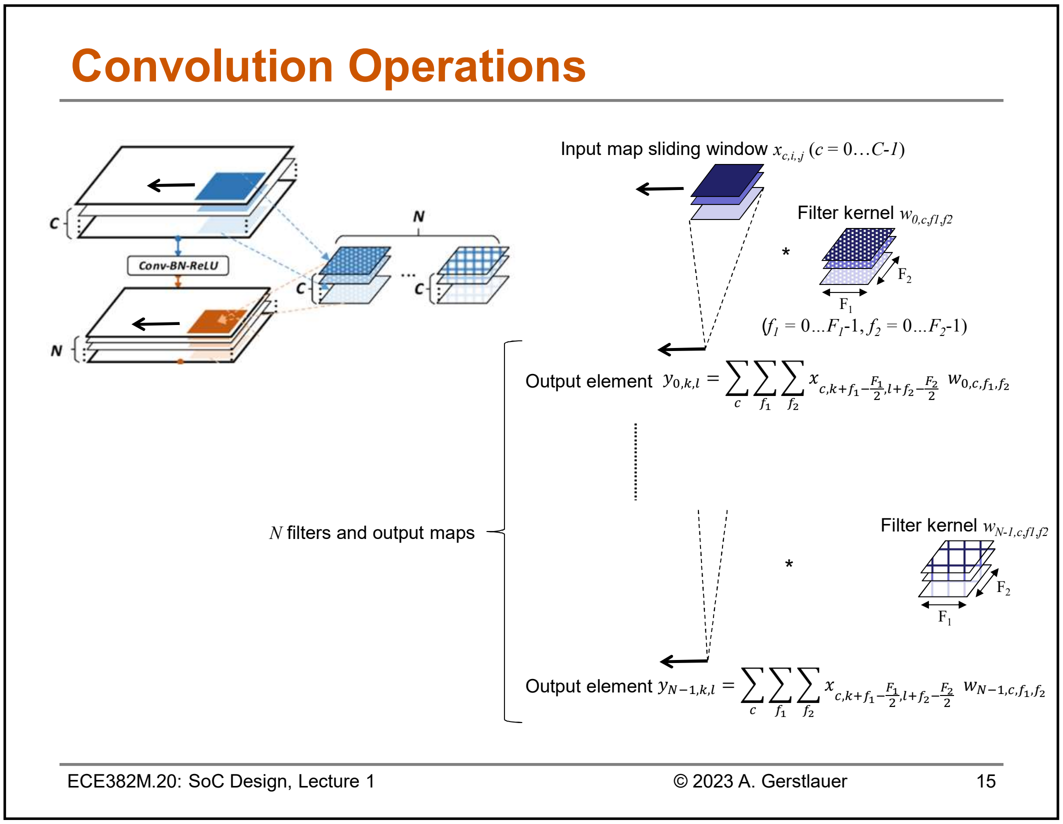


Digit recognition CNN (image classification)

➢ **Fully convolutional network (FCN): no fully connected layer**

Convolution Operations





Convolution Operations

Input map sliding window $x_{c,i,j}$ $(c = 0 \ldots C\text{-}1)$

Filter kernel $w_{0,c,f1,f2}$

$*$

F_1 F_2

$(f_1 = 0 \ldots F_1\text{-}1, f_2 = 0 \ldots F_2\text{-}1)$

Output element $y_{0,k,l} = \sum_{c} \sum_{f_1} \sum_{f_2} x_{c,k+f_1-\frac{F_1}{2},l+f_2-\frac{F_2}{2}} \, w_{0,c,f_1,f_2}$

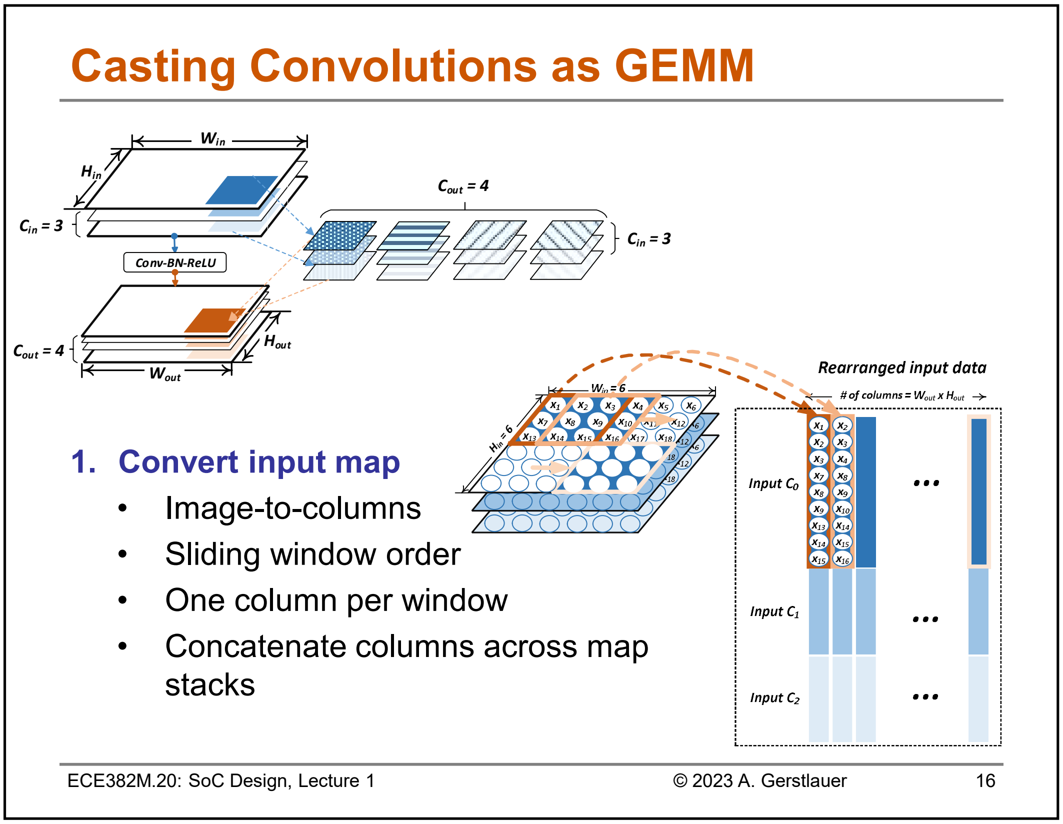
N filters and output maps

Filter kernel $w_{N\text{-}1,c,f1,f2}$

$*$

F_1 F_2

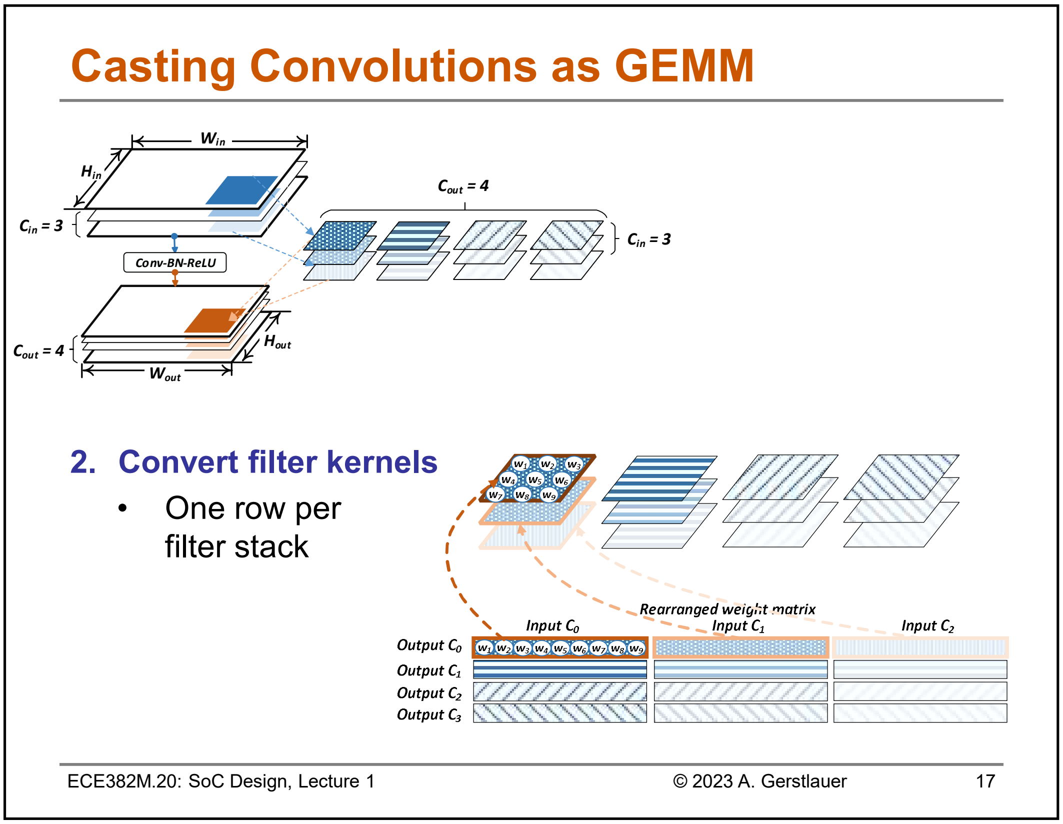
Output element $y_{N-1,k,l} = \sum_{c} \sum_{f_1} \sum_{f_2} x_{c,k+f_1-\frac{F_1}{2},l+f_2-\frac{F_2}{2}} \, w_{N-1,c,f_1,f_2}$



Casting Convolutions as GEMM

W_{in}
H_{in}
$C_{in} = 3$
Conv-BN-ReLU
$C_{out} = 4$
$C_{in} = 3$
$C_{out} = 4$
H_{out}
W_{out}
$W_{in} = 6$
$H_{in} = 6$
x_1 x_2 x_3 x_4 x_5 x_6
x_7 x_8 x_9 x_{10} x_{11} x_{12}
x_{13} x_{14} x_{15} x_{16} x_{17} x_{18}
Rearranged input data
# of columns = W_{out} x H_{out}
Input C_0
x_1 x_2 x_3 x_7 x_8 x_9 x_{13} x_{14} x_{15}
x_2 x_3 x_4 x_8 x_9 x_{10} x_{14} x_{15} x_{16}
...
Input C_1
...
Input C_2
...

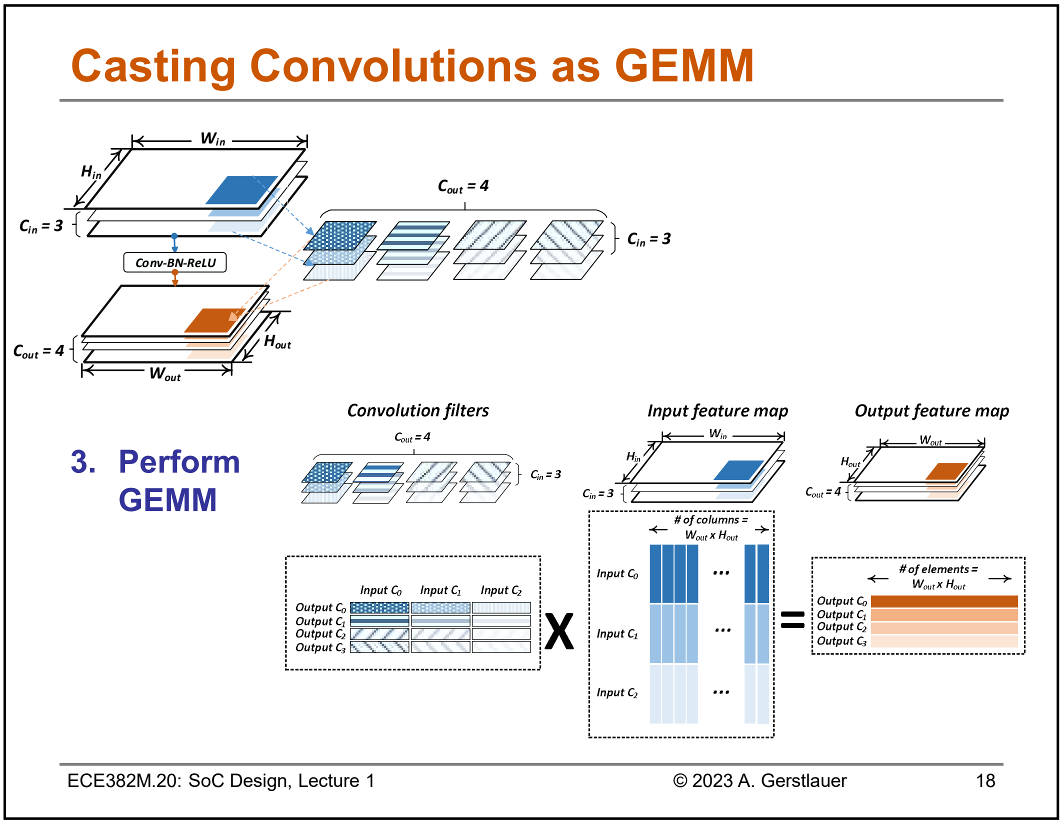
1. **Convert input map**
   - Image-to-columns
   - Sliding window order
   - One column per window
   - Concatenate columns across map stacks



Casting Convolutions as GEMM

2. **Convert filter kernels**
   - One row per filter stack

ECE382M.20: SoC Design, Lecture 1 © 2023 A. Gerstlauer 17



Casting Convolutions as GEMM

3. **Perform GEMM**

ECE382M.20: SoC Design, Lecture 1 © 2023 A. Gerstlauer 18

Project Description

- **HW/SW co-design of an embedded SoC**
  - Low-power YOLO/Darknet implementation
  - ARM-based target platform
    - ARM Cortex-A9 processor, memory components, I/O devices
    - Custom hardware accelerators (GEMM)
    - Interconnected via standard system busses or memory/cache interfaces
  - Virtual and physical prototyping
    - SystemC TLM-based virtual platform model (QEMU ARM simulator)
    - ARM- and Xilinx FPGA-based prototyping board (Zynq-7000)

  ➢ Lab and project in teams

    ➢ 2-3 per team, 20 teams for 20 boards

Project Objectives and Activities

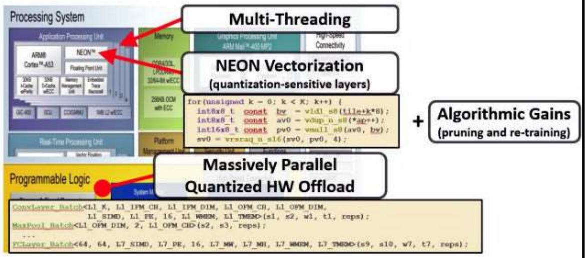
- **Project objective:**
  - Implement the YOLO/Darknet code on a ARM based SoC while meeting the performance, area and power metrics.
- **Project activities:**
  - Profile the YOLO/Darknet software implementation to determine performance bottlenecks
  - Optimize the YOLO/Darknet software (fixed point operation)
  - Partition the software into components which will run on the ARM processor and on hardware accelerators
  - Synthesize accelerators into Verilog for gate level implementation
  - Co-simulate and prototype the HW/SW implementation
  - Estimate timing, area and power metrics and validate against product requirements

Development Tasks

- **ARM software development**
  - Compile and profile YOLO/Darknet on ARM board
  - Convert floating-point to fixed-point code and check mAP
  - Compile and profile fixed-point Yolo on ARM board
  - Optimize software on dual-core ARM platform
  - Develop hardware abstraction layer (HAL) and I/O handler
  - Develop interrupt handler & driver (Linux kernel module)
- **Hardware development on FPGA**
  - Hardware accelerators (synthesize fixed-point code)
  - Interface to ARM board and on-chip bus
  - Memory/cache interfaces (optional DRAM controller)
  - Interrupt logic, clocking & reset
  - Debug, diagnostics

Xilinx Tincy YOLO on Zynq

TincyYOLO demo at NIPS'17

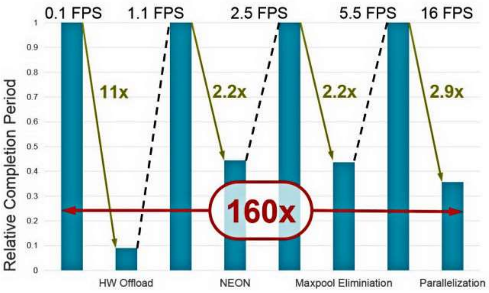


```
for(unsigned k = 0; k < K; k++) {
  int8x8_t  const  bv  = vld1_s8(tile+k*8);
  int8x8_t  const  av0 = vdup_n_s8(*ap++);
  int16x8_t const  pv0 = vmull_s8(av0, bv);
  sv0 = vrsraq_n_s16(sv0, pv0, 4);
```

```
ConvLayer_Batch<L1_K, L1_IFM_CH, L1_IFM_DIM, L1_OFM_CH, L1_OFM_DIM,
                L1_SIMD, L1_PE, 16, L1_WMEM, L1_TMEM>(s1, s2, w1, t1, reps);
MaxPool_Batch<L1_OFM_DIM, 2, L1_OFM_CH>(s2, s3, reps);
  ...
FCLayer_Batch<64, 64, L7_SIMD, L7_PE, 16, L7_MW, L7_MH, L7_WMEM, L7_TMEM>(s9, s10, w7, t7, reps);
```

- **Starting from Tiny YOLO**
  - Smaller CNN model for constrained devices
- **HW/SW co-optimizations**
  - HW acceleration
  - SW optimizations

https://t.co/ffkZgMRmwM


0.1 FPS
1.1 FPS
2.5 FPS
5.5 FPS
16 FPS
11x
2.2x
2.2x
2.9x
160x
Relative Completion Period
HW Offload
NEON
Maxpool Eliminiation
Parallelization