MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints

Seungyeop Han[1], Haichen Shen[1], Matthai Philipose[2], Sharad Agarwal[2], Alec Wolman[2], Arvind Krishnamurthy[1]

1: University of Washington
2: Microsoft Research

Presented by:
Cody LaFlamme, Michael Wollenhaup

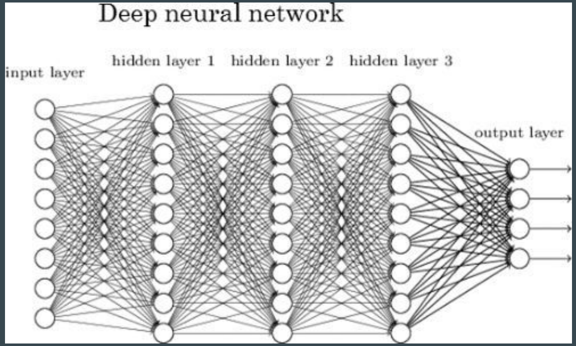
https://homes.cs.washington.edu/~arvind/papers/mcdnn.pdf

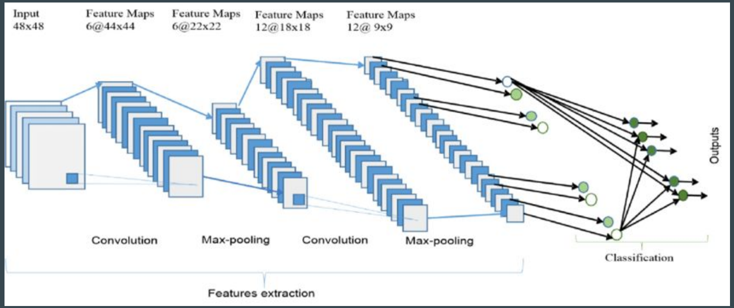
What are DNN?

- A **Deep Neural Network** is classified by it's large amount of hidden layers. A neural network is considered "deep" if it's **Credit Assignment Path** (CAP) index is greater than 2

-Rather than layers in a neuron layout, DNN (specifically CNN) can be represented by an array (the input image) where each layer is a matrix operation. These operations include:

-Matrix Multiplication
-Convolution
-Max-Pooling
-Non-Linearizing
-Rescaling





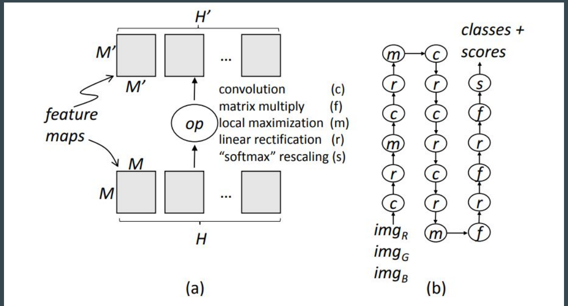


Figure 2: (a) DNN "layers" are array operations on lists of arrays called feature maps. (b) A state-of-the-art network for scene recognition, formed by connecting layers.

The Resources Required for DNN

-DNN (Deep Neural Networks) are the dominant approach, especially in computer vision applications, due to their "excellent recognition performance"

-However, DNN are expensive in terms of memory and GFLOPS of computing power required for an operation, so they are typically only used in server based scenarios

-Due to the power of DNN, and the applications of DNN, there is strong motivation for a mobile implementation

-There are some solutions currently in place such as hand-crafted DNN for execution on co-processors (sw) and custom hardware accelerators (hw)

	face [46]	scene [52]	object [44]
training time (days)	3	6	14-28
memory (floats)	103M	76M	138M
compute (FLOPs)	1.00G	2.54G	30.9G
accuracy (%)	97	51	94

Table 1: Although DNNs deliver state-of-the art classification accuracy on many recognition tasks, this functionality comes at very high memory (space shown is to store an active model) and computational cost (to execute a model on a *single* image window).

Continuous Mobile Vision Systems

-This paper will specifically target enabling a large collection of face, scene, and object recognition algorithms to be applied to video streams from mobile devices

-The DNN are expected to be run multiple times to many of the frames

-The example to the right is a “state-of-the-art mobile/cloud Continuous Mobile Vision (CMV) system

-A resolution of 4096x2160 pixels (large FOV) @ 15 FPS would be a reasonable use-case of this model

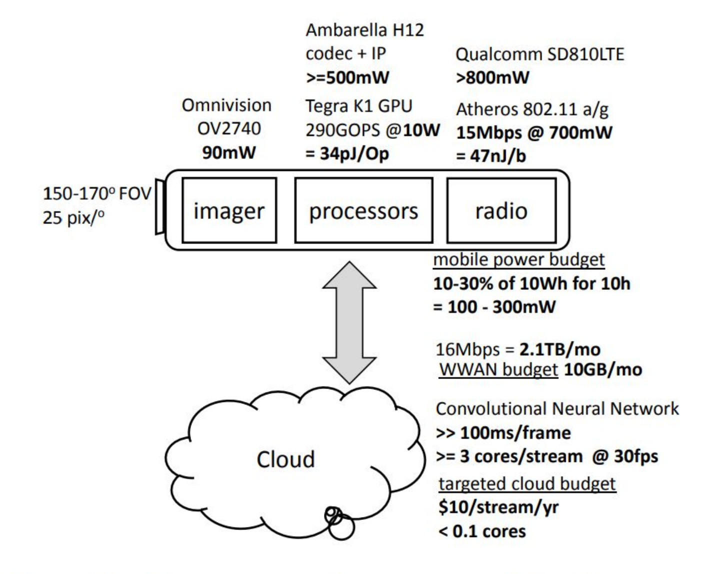


Figure 1: Basic components of a continuous mobile vision system.

Continuous Mobile Vision Systems

-The device would have an expected power consumption of around 1.3-1.6 W (90 mW for imaging, 0.7-1 W for wireless offload, and 0.5 W on compression)
-Utilizing a realistic compression (100x) would yield a 16 Mbps stream, or around 2.1 TB per month at 10 hours of usage per day
-If we make a consertive estimation of 1 DNN applied per frame and 100 ms execution latency; a single CPU would need 1.5-3 cores (commonly more) to keep up with 15-30 frames per second
-However, a large 3Ah mobile phone battery would yield roughly 1.2 W over 10 hours; a mobile plan would only allow around 10 GB per month

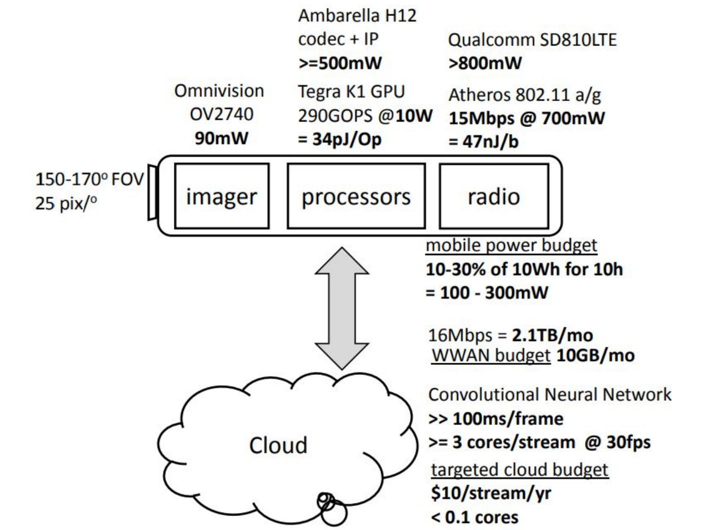
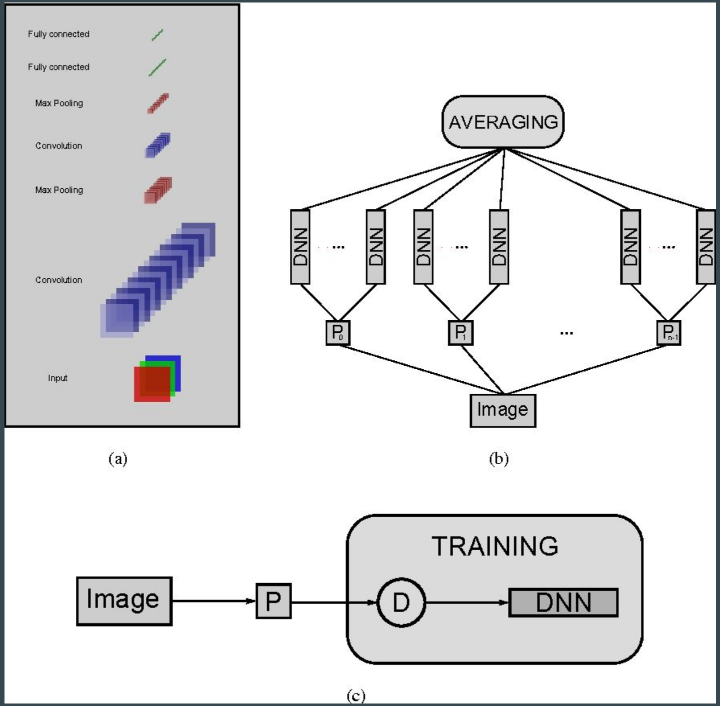


Figure 1: Basic components of a continuous mobile vision system.

Approximation as a Solution for CMVS

-This paper implements a solution to this problem that finds itself part way between the hardware accelerators and the handcrafted DNN software solution. It is called model optimization.
-These are techniques that apply automatically to any DNN and reduce associated memory and processing costs, typically at the cost of classification accuracy.
-It is possible to sacrifice a moderate amount of accuracy (1%-3%), while obtaining memory reduction that allow the models to fit within mobile memories (10x reduction), and processing reduction that allows the model to be executed on a mobile GPU (3x reduction)



(a)
(b)
(c)

How will the DNN be optimized

-When optimizing we are willing to sacrifice accuracy for decreased computations and storage

-The figure on the right displays that the resource distribution is largely in-balanced in regards to output size relative to the resources used (storage and computations)

There are 3 main methods for optimizing, used in this paper:

-Matrix Factorization

-Matrix Pruning

-Architecture Changes

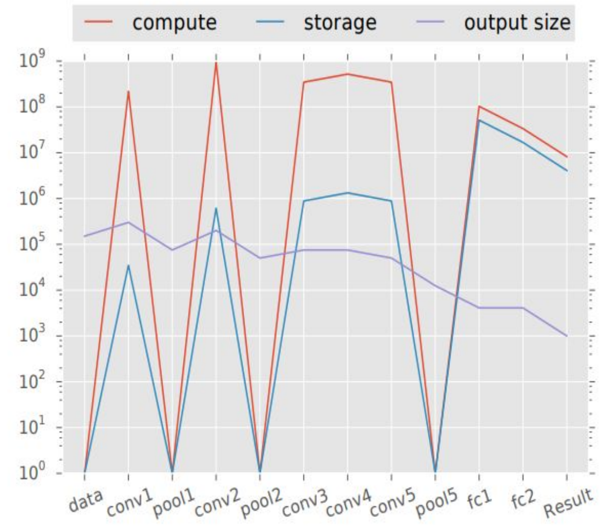


Figure 3: Resource usage of AlexNet across layers (note log scale).

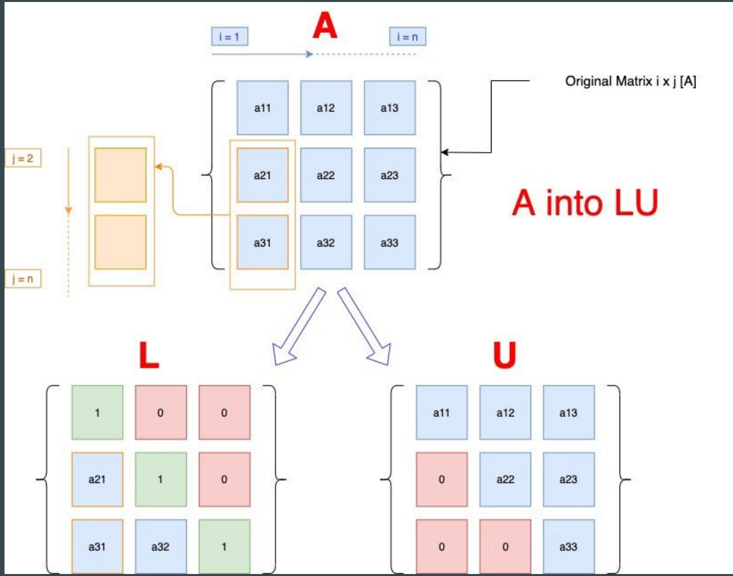
Optimization 1: Matrix Factorization

- "Matrix Factorization replaces the weight matrices and convolution kernels by their low rank approximations"

Benefits:

-Replacing a M x M weight matrix W(M x M) with its singular value decomposition U(M x k)V(k x M) reduces storage overhead from M^2 to 2Mk and computational overhead from M^3 to 2M^(2)k.

-Recent results report 5.5x memory use reductions and 2.7x FLOP reduction, at a loss of only 1.7% accuracy for the AlexNet model, and 1.2x and 4.9x reductions for a 0.5% accuracy loss



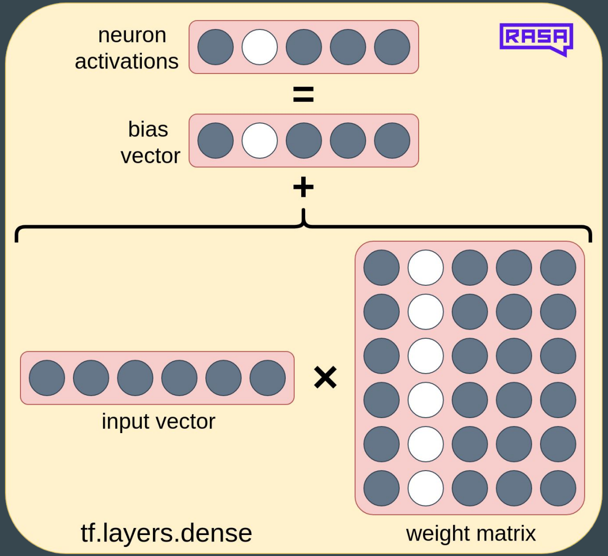
Optimization 2: Matrix Pruning

-"Matrix pruning sparsifies matrices by zeroing very small values, use low-bitwidth representations for remaining non-zero values and use compressed representations for those values."

Benefits:

-The most recent results report a 11/8x reduction in model size and a 3/5x reduction on FLOPs for AlexNet/VGGNet, while sacrificing essentially no accuracy

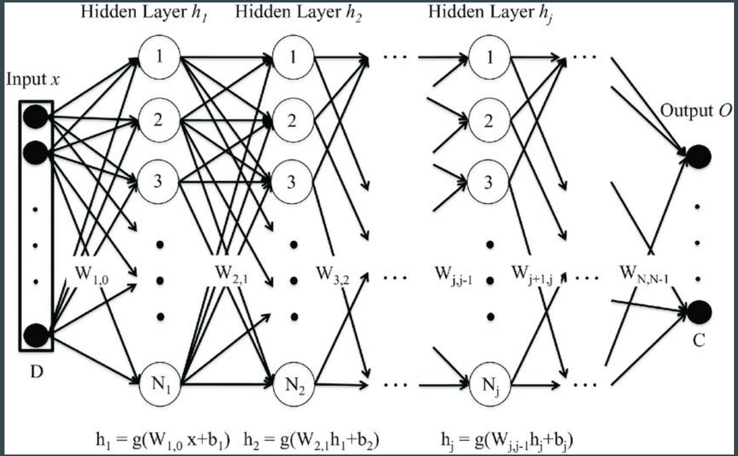
-However, giving up 2% of accuracy can improve memory size reduction to 20x



Optimization 3: Architecture Changes

-"Architectural changes explore the space of model architectures, including varying the number of layers, size of weight matrices including kernels, etc."

-For example, reducing the number of layers in a DNN from 19 to 11 results in a drop of accuracy of 4.1%



Approximation Results: Memory

Note the log-scale Y axis!

Task	**Description** (# training images, # test images, # class)
V	VGGNet [44] on ImageNet data
A	AlexNet on ImageNet data [18] for object recognition (1.28M, 50K, 1000)
S	AlexNet on MITPlaces205 data [52] for scene recognition (2.45M, 20K, 205)
M	re-labeled S for inferring manmade/natural scenes
L	re-labeled S for inferring natural/aritificially lighting scenes
H	re-labeled S with Sun405 [49] for detecting horizons
D	DeepFaceNet replicating [46] with web-crawled face data (50K, 5K, 200)
Y	re-labeled D for age: 0-30, 30-60, 60+
G	re-labeled D for gender: M, F
R	re-labeled D for race: African American, White, Hispanic, East Asian, South Asian, Other

Table 2: Description of classification tasks.

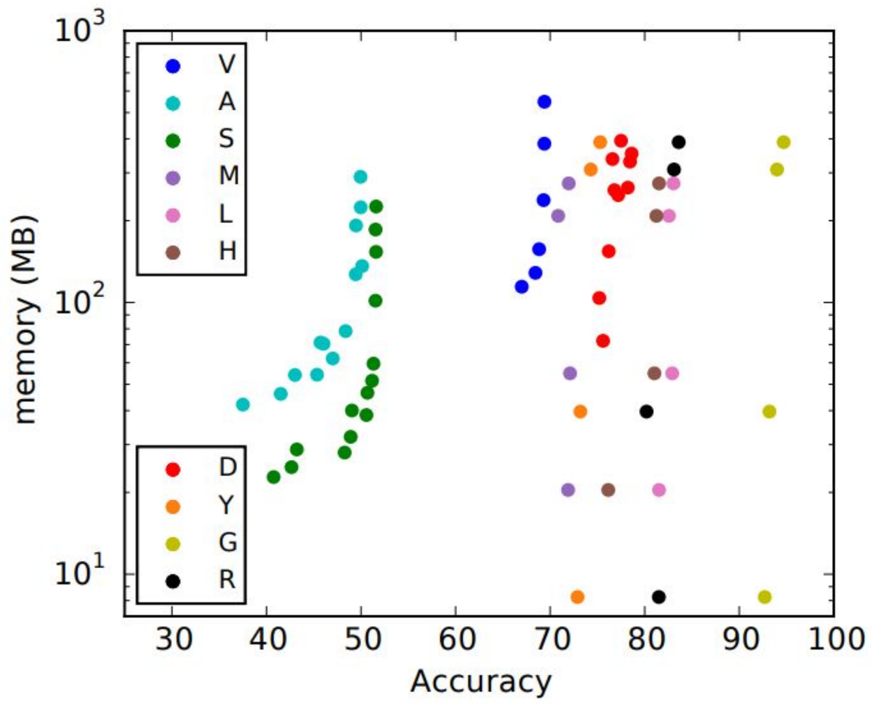


Figure 4: Memory/accuracy tradeoffs in MCDNN catalogs.

Approximation Results: Energy

Y-axis is *not* log-scale here.

Task	**Description** (# training images, # test images, # class)
V	VGGNet [44] on ImageNet data
A	AlexNet on ImageNet data [18] for object recognition (1.28M, 50K, 1000)
S	AlexNet on MITPlaces205 data [52] for scene recognition (2.45M, 20K, 205)
M	re-labeled S for inferring manmade/natural scenes
L	re-labeled S for inferring natural/aritificially lighting scenes
H	re-labeled S with Sun405 [49] for detecting horizons
D	DeepFaceNet replicating [46] with web-crawled face data (50K, 5K, 200)
Y	re-labeled D for age: 0-30, 30-60, 60+
G	re-labeled D for gender: M, F
R	re-labeled D for race: African American, White, Hispanic, East Asian, South Asian, Other

Table 2: Description of classification tasks.

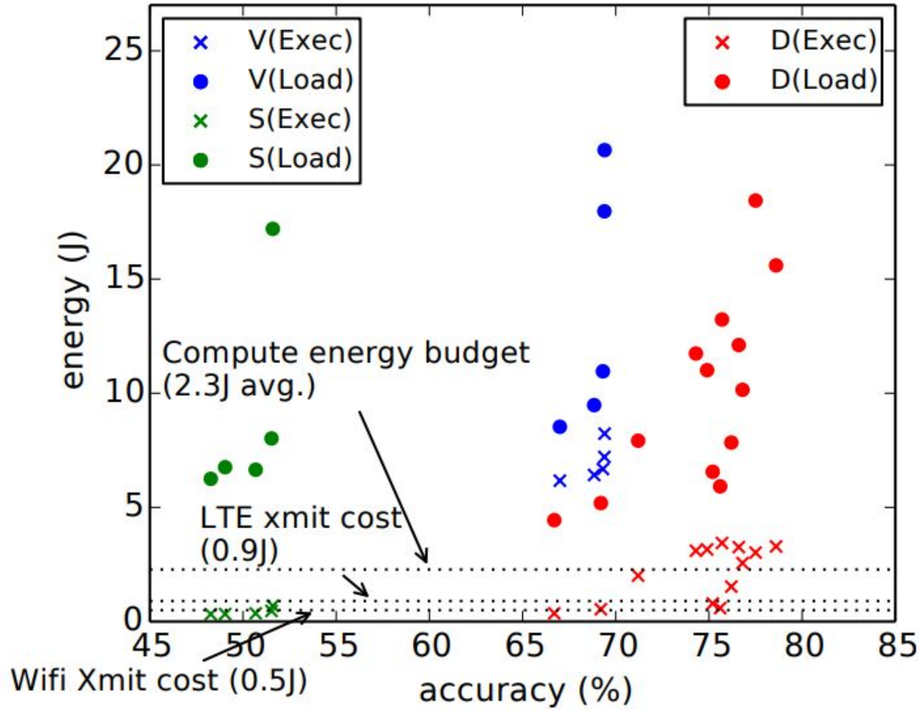


Figure 5: Energy/accuracy tradeoffs in MCDNN catalogs.

Approximation Results: Latency

Back to a log-scale Y-axis!

Task	Description (# training images, # test images, # class)
V	VGGNet [44] on ImageNet data
A	AlexNet on ImageNet data [18] for object recognition (1.28M, 50K, 1000)
S	AlexNet on MITPlaces205 data [52] for scene recognition (2.45M, 20K, 205)
M	re-labeled S for inferring manmade/natural scenes
L	re-labeled S for inferring natural/aritificially lighting scenes
H	re-labeled S with Sun405 [49] for detecting horizons
D	DeepFaceNet replicating [46] with web-crawled face data (50K, 5K, 200)
Y	re-labeled D for age: 0-30, 30-60, 60+
G	re-labeled D for gender: M, F
R	re-labeled D for race: African American, White, Hispanic, East Asian, South Asian, Other

Table 2: Description of classification tasks.

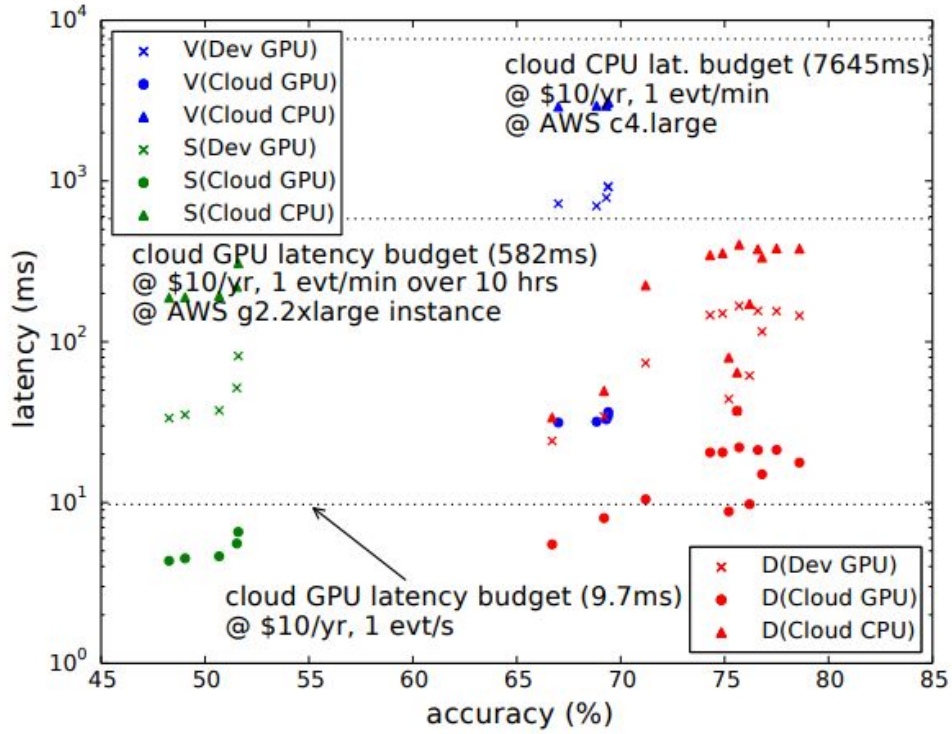


Figure 6: Latency/accuracy tradeoffs in MCDNN catalogs.

AMS: Approximate Model Scheduling

- We've established an accuracy/resources tradeoff...
  - **How much do we approximate?**
- Maximize accuracy while under **constraints**:
  - Energy consumption
  - Cloud utilization
  - Cache/Memory capacity
  - Time (limited by use case)
    - Latency (input -> output delay)
    - Framerate (output frequency)
- **Similar to knapsack/packing problem!**
  - Create large "catalogue" of "variant" networks
  - Choose optimal set of approximate models while staying under constraints
- **Major difference: constraints can change over time!**

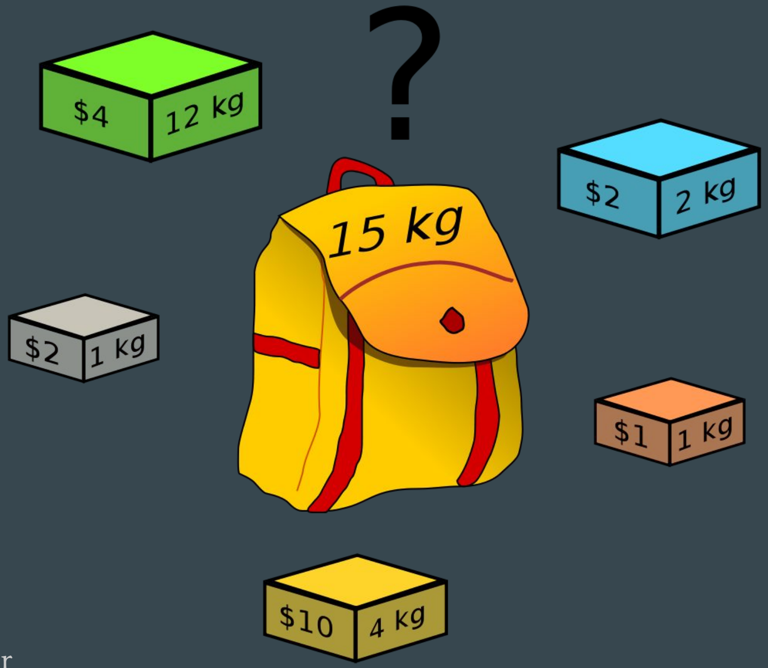


Figure:

Novel Approximation: Specialization

- A model might recognize thousands of people...
- But we don't usually see that many people at once!
- Dynamically retrain models that specialize in recognizing what is being observed!
    - Need to save some time, though...
        - **Pretarget**: train only output layer!
        - **Pre-forward**: store output from second-last layer as input for the output layer!

Task (variant)	**Time to specialize (s)**		
	Full re-train	**+ Retarget**	**+ Pre-forward**
Face (C0)	2.6e4	30.4	4.3
Face (C4)	1.4e4	24.0	4.2
Object (A0)	4.8e5	152.4	14.2
Object (A9)	9.1e4	123.0	14.1

Table 3: Runtime overhead of specialization.

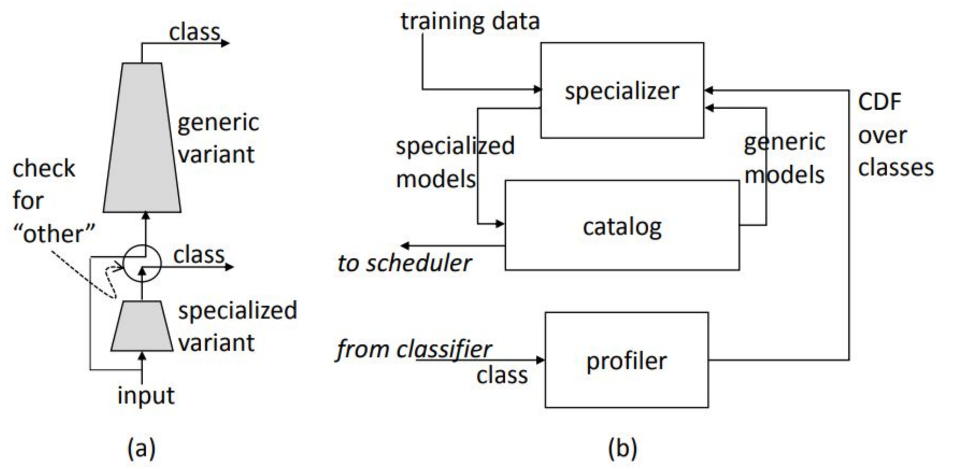


Figure 7: Model specialization: (a) Cascading specialized models. (b) MCDNN infrastructure for specialization.

Novel Approximation: Sharing

- A neural network transforms input data into data of a different type.
  - At output, this can be classifications or predictions...
  - In the middle layers, the data still means something!
    - Middle-layer output can be considered a unique encoding of the input.
- **Different networks that recognize facial features might have very similar early layers!**
  - Train early layers and late layers separately...
  - Make similar networks **share** the same early layers...
  - **Only load/run the early layers once!**

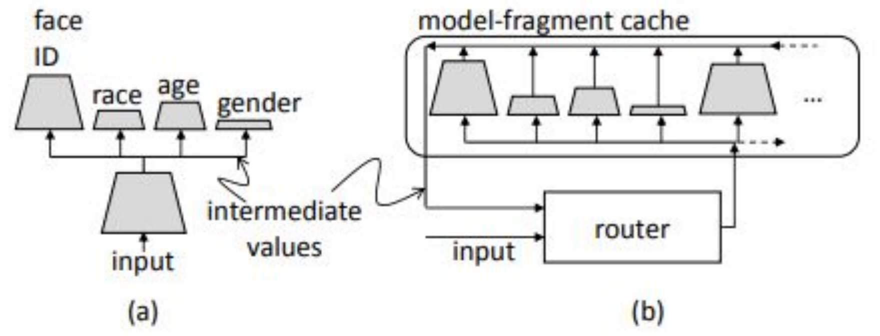
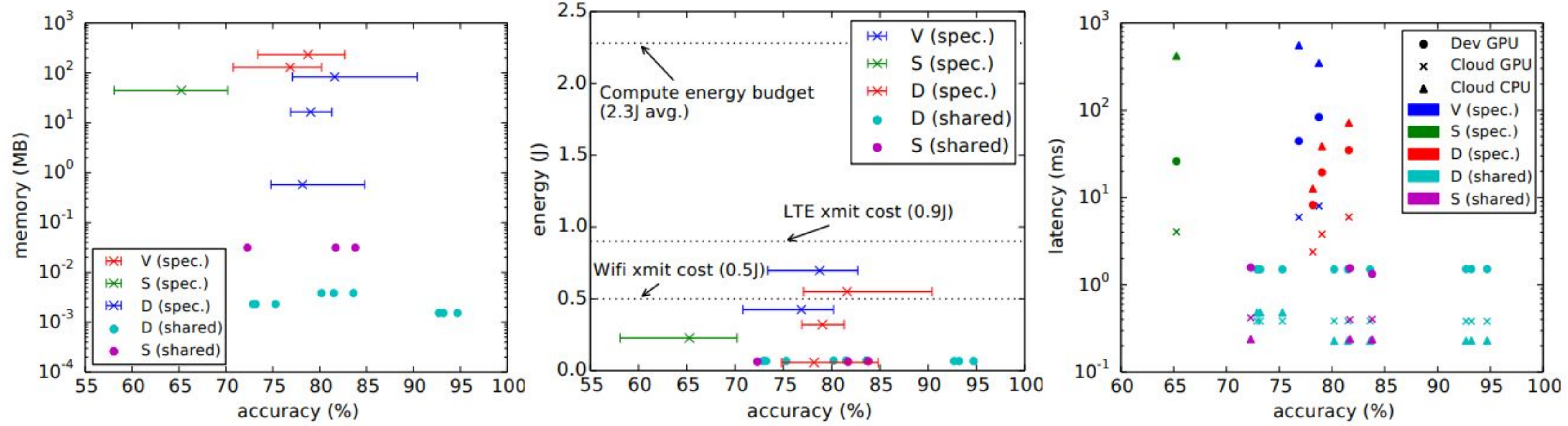


Figure 8: Model sharing: (a) Sharing model fragments for facial analysis. (b) MCDNN infrastructure for sharing, replicated in the client and cloud.

Effects of Specialization / Sharing

Reduces resource use across the board, and doesn't hurt accuracy too much.

Importantly: We meet our energy constraint now!



(a) Memory consumption **(b)** Energy consumption **(c)** Execution latency

Figure 10: Impact of collective optimization (best viewed in color).

Example Results!

- **"Original Model:"**
    - No optimization/approximation
- **"Best Model:"**
    - Static approximated model
    - Chosen from "knee" of approximation curves
    - *Not* dynamically scheduled as constraints change
    - *Does not* use specialization or sharing
- **"All Models:"**
    - **The paper's proposed solution.**
    - Models generated using all discussed methods.
    - Scheduled using MCDNN.
    - **Meets demands with good accuracy :)**

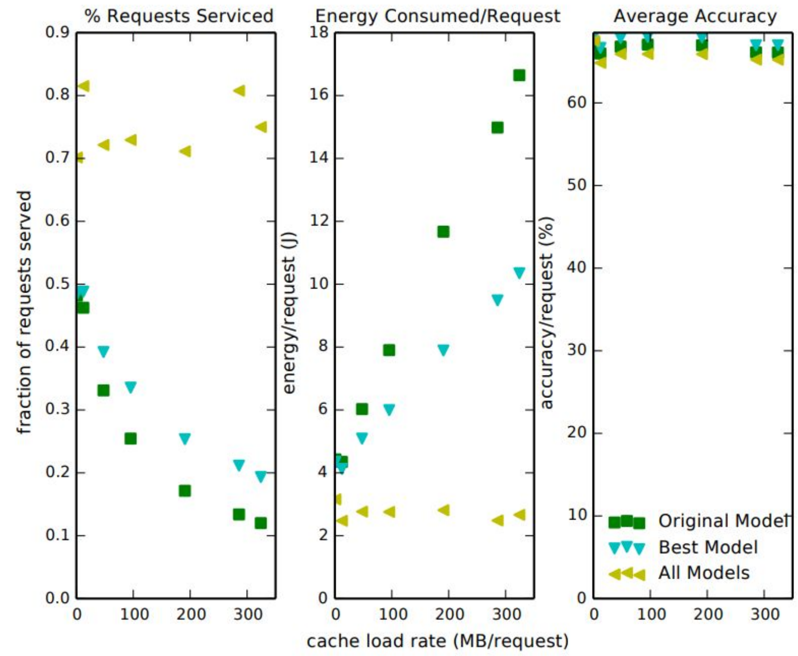


Figure 11: Impact of MCDNN's dynamically-sized caching scheme.

More Example Results!

- Ran several computer vision applications at once for a day on a mobile device.
- The cloud is still needed to make it through the day.
- Spotty cloud connection is acceptable.

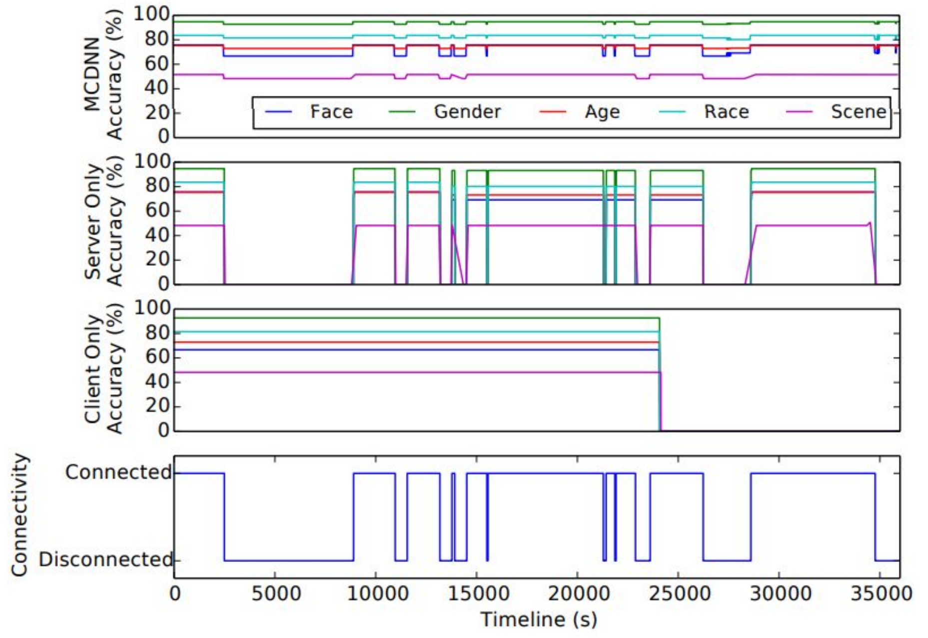


Figure 13: Accuracy of each application over time for the Glimpse usage. Each line shows a single application.

Architecture Recap/Overview

- **Compile/Training time:**
  - Programmer supplies:
    - Input type (e.g. faces)
    - Model schema (e.g. 8 layer CNN)
    - Training data
    - Validation data
  - Compiler creates:
    - Fully trained models
    - Catalog of approximated models
- **Runtime, at every time step**:
  - Specializer specializes, if possible :)
  - Scheduler dictates execution:
    - Chooses model
    - Chooses where to execute it
  - Models all execute
  - Output is returned to user

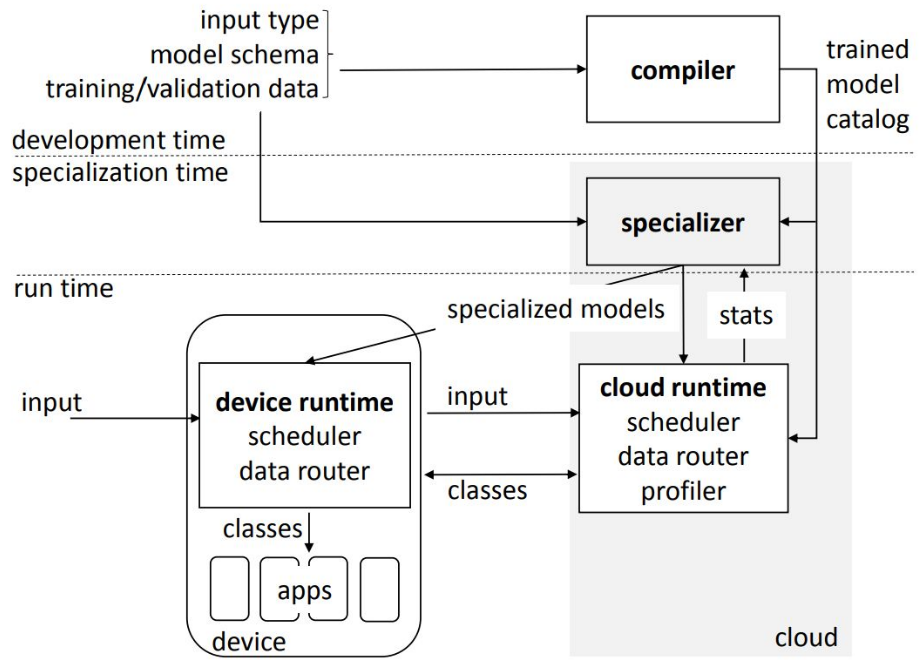


Figure 9: Architecture of the MCDNN system.

Questions?

Multiple Choice Q’s (do we even need these?)

Paper: “MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints”

1. Select from the following all that can be used to create an approximate version of a neural network:
   a. **Matrix Pruning**
   b. **Matrix Factorization**
   c. Matrix Inversion
   d. **Architecture Changes**
   e. Adam Optimization

2. Which one of the following is the *largest* obstacle for deep neural networks running on mobile devices (with current technology)?
   a. Memory use
   b. **Energy use**
   c. Weak processing
   d. Spotty network connectivity

3. In this paper, “specialization” of neural networks is made possible due to which one of the following:
   a. Pre-training many smaller neural networks using subsets of the training data
   b. Training the network as it is executing, using recent outputs as training data
   c. Sacrificing flexibility and accepting that the system is only *ever* going to be used in a small number of situations
   d. **Re-training the output layer at runtime on a subset of the training data**