Accelerating Deep Convolution Neural Networks for Large-Scale Speech Tasks using GPUs

Ewout van den Berg, Daniel Brand*, Leonid Rachevsky, Rajesh Bordawekar*, Bhuvana Rambhadran

Watson Multimodal and *IBM Research

3/17/2015

Outline

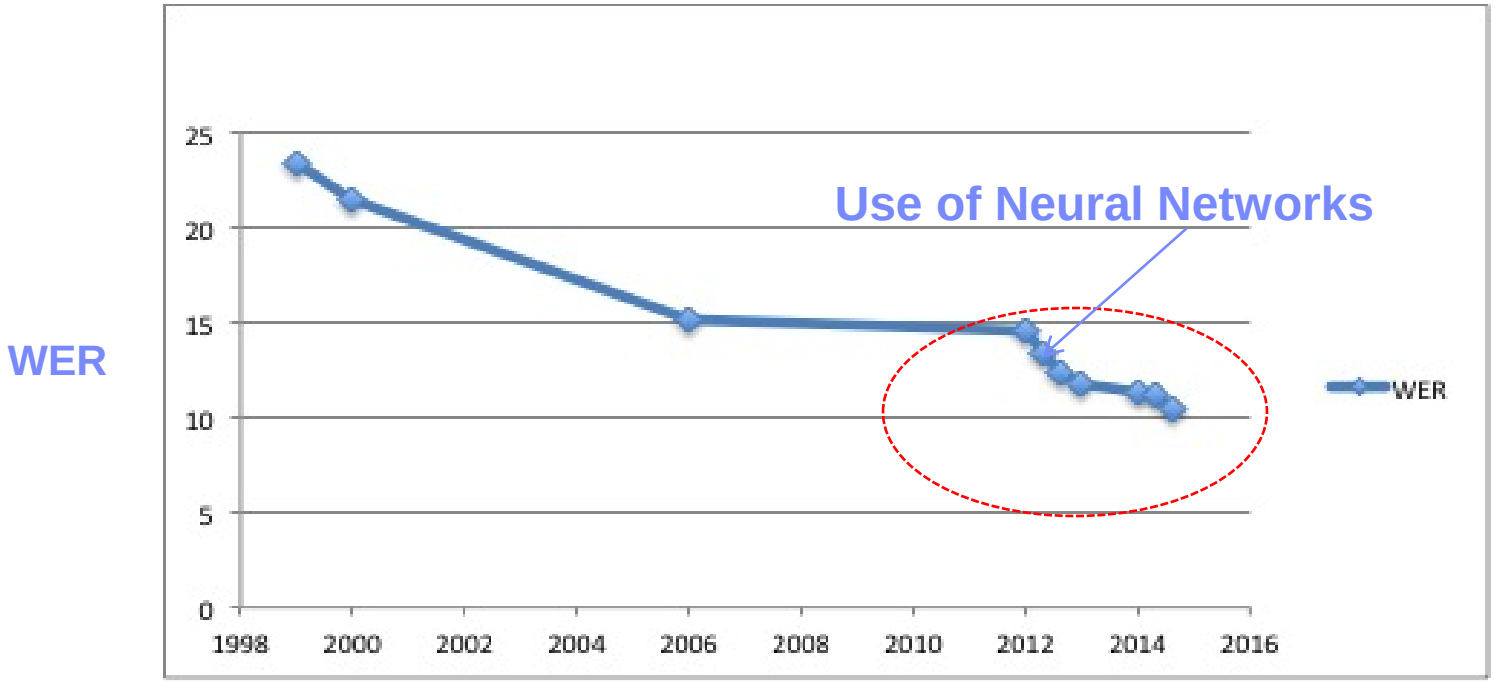
- Deep Learning for Speech Processing
- IBM Speech Processing System
  - Overall System Design
  - **Implementation of the Convolution Layers**
- Performance Evaluation of Convolution Layers
  - Native, CUDNN, and CUFFT
- Conclusions and Future Work

Automatic speech recognition

- Goal: Convert audio to transcribed text
  - maximize probability p(sequence of words | input data)
- Ingredients:
  - Dictionary: words as sequence of phonemes
  - Acoustic model: p(phoneme | data segment)
  - Language model: p(word | previous words)
  - Maximization based on HMM and Viterbi decoding
- This presentation: **focus on acoustic model**

Neural networks in acoustic models

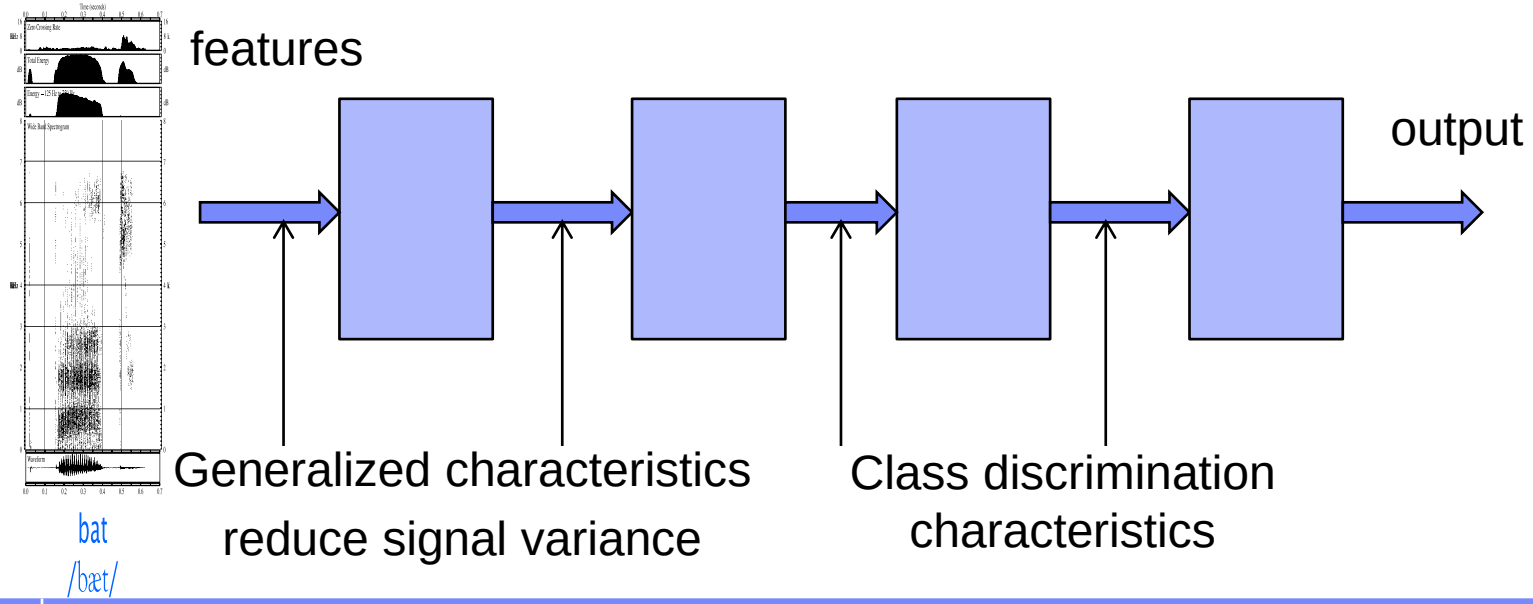
- Traditionally GMMs were used
- Neural networks greatly improved performance



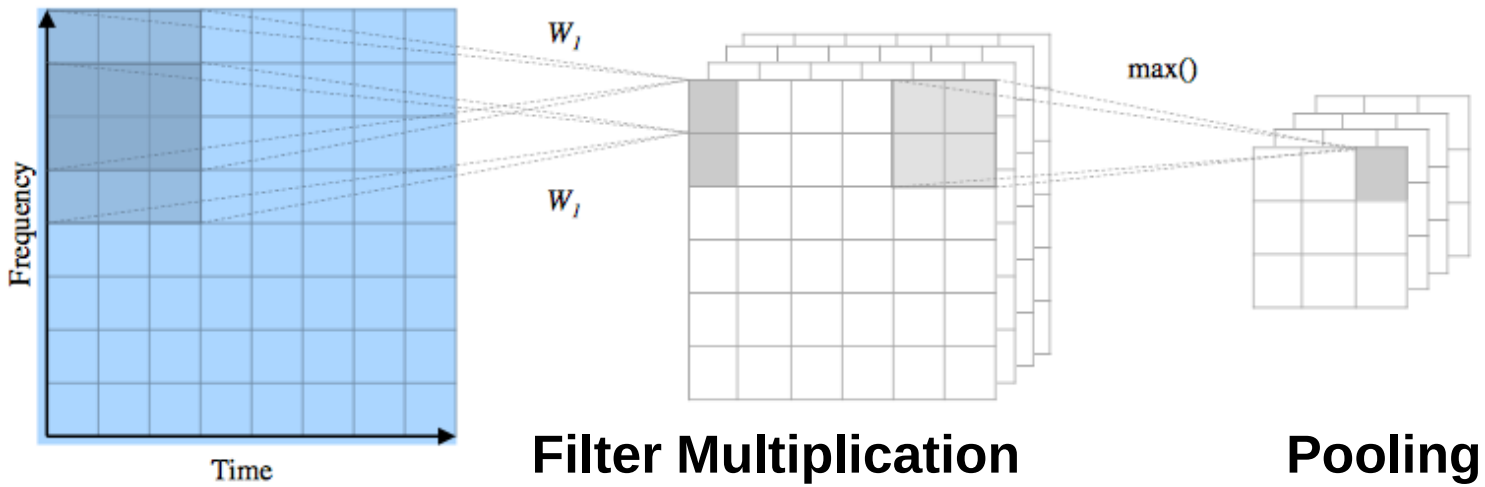
- Very active area of research: specific focus on exploitation of deep (DNN) and convolution neural networks (CNN)

Multi-Layer DNN architecture for speech processing

- Input audio data is transformed into simple feature vectors
- Neural network layers transform data to obtain higher-order, more complex representations of the input
  - each layer produces a higher-level feature representation and better classifier than its input
- By combining simple layers, we can design more complex systems
- For acoustic models, output is normalized to obtain probability p(phoneme | feature)



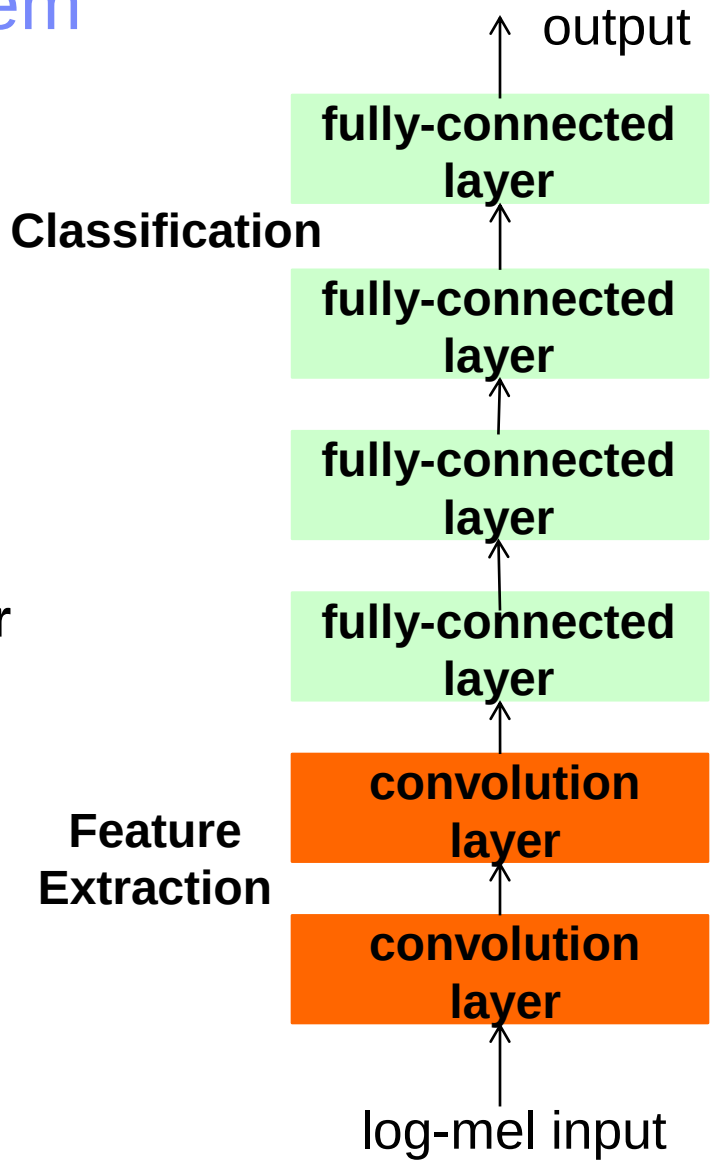
Convolutional Neural Network Architecture



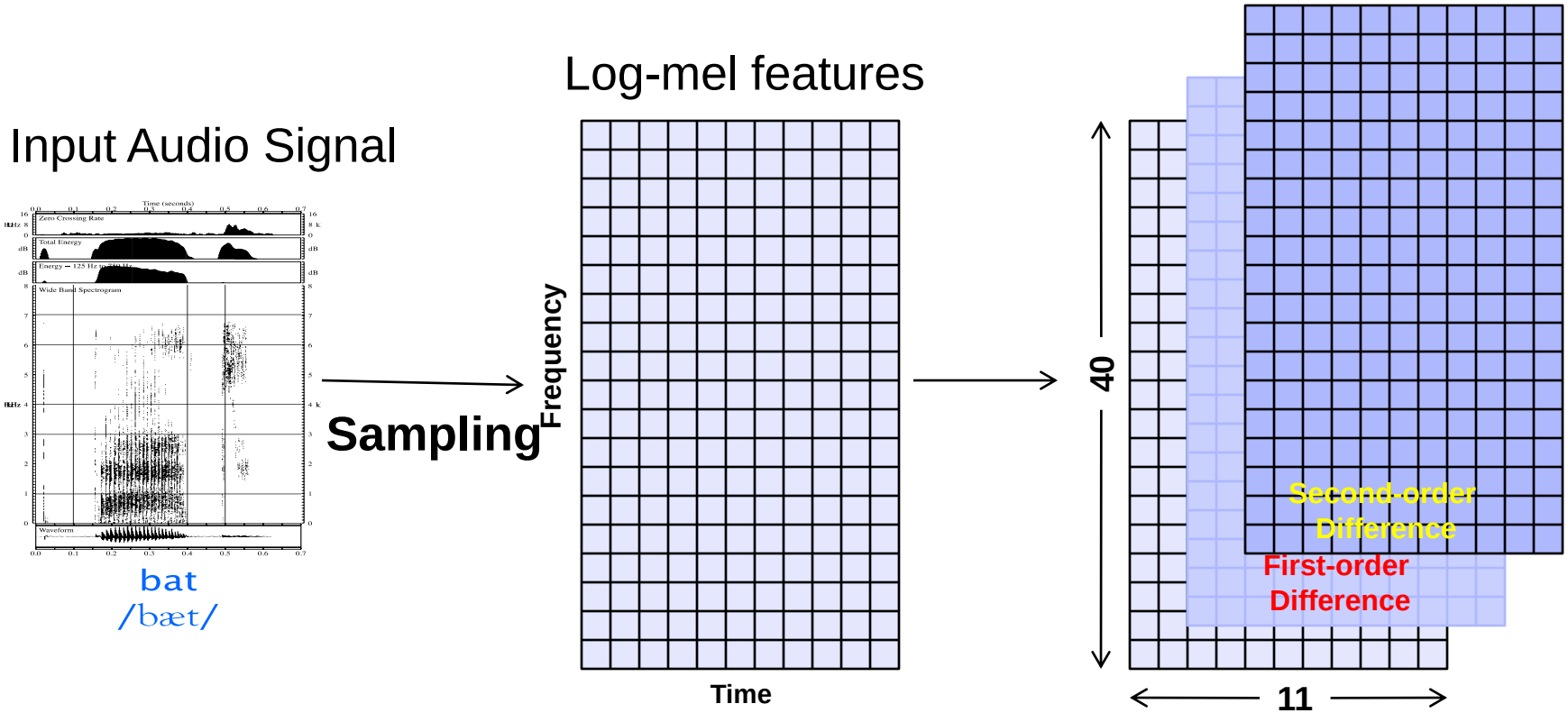
- Designed to exploit 2-dimensional representations of input data
- CNNs can model translation invariances in audio (e.g. subtle differences in pronunciation)
- A full CNN weight layer consists of
  - Locally connected filter multiplications
  - Pooling the outputs after multiplication

IBM Speech Processing System

- **Uses both CNNs and DNNs**
- **CNNs can only be used for features that obey locality in time and frequency (log-mel)**
- **2 convolution layers appropriate for speech processing**
- **DNNs are used for classification**
- **Entire network trained jointly**

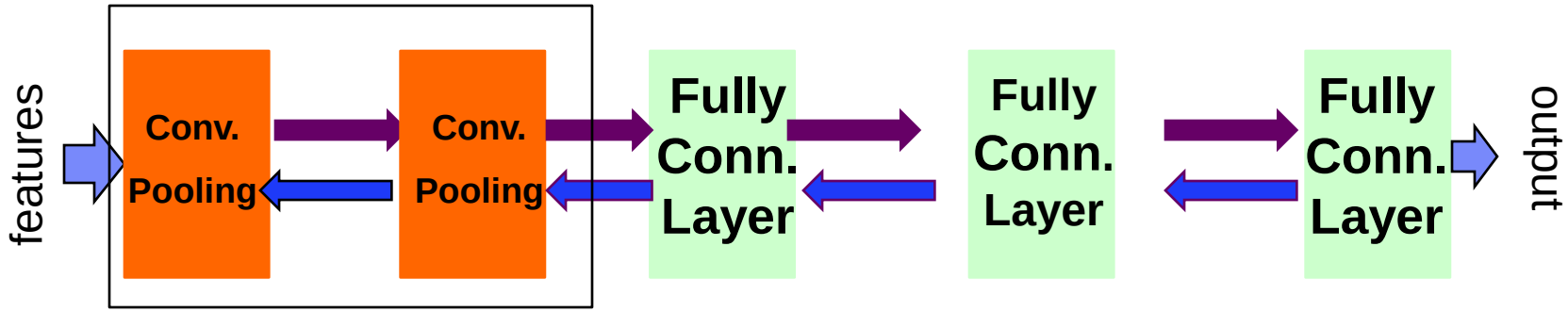


Input for the IBM Speech Processing System



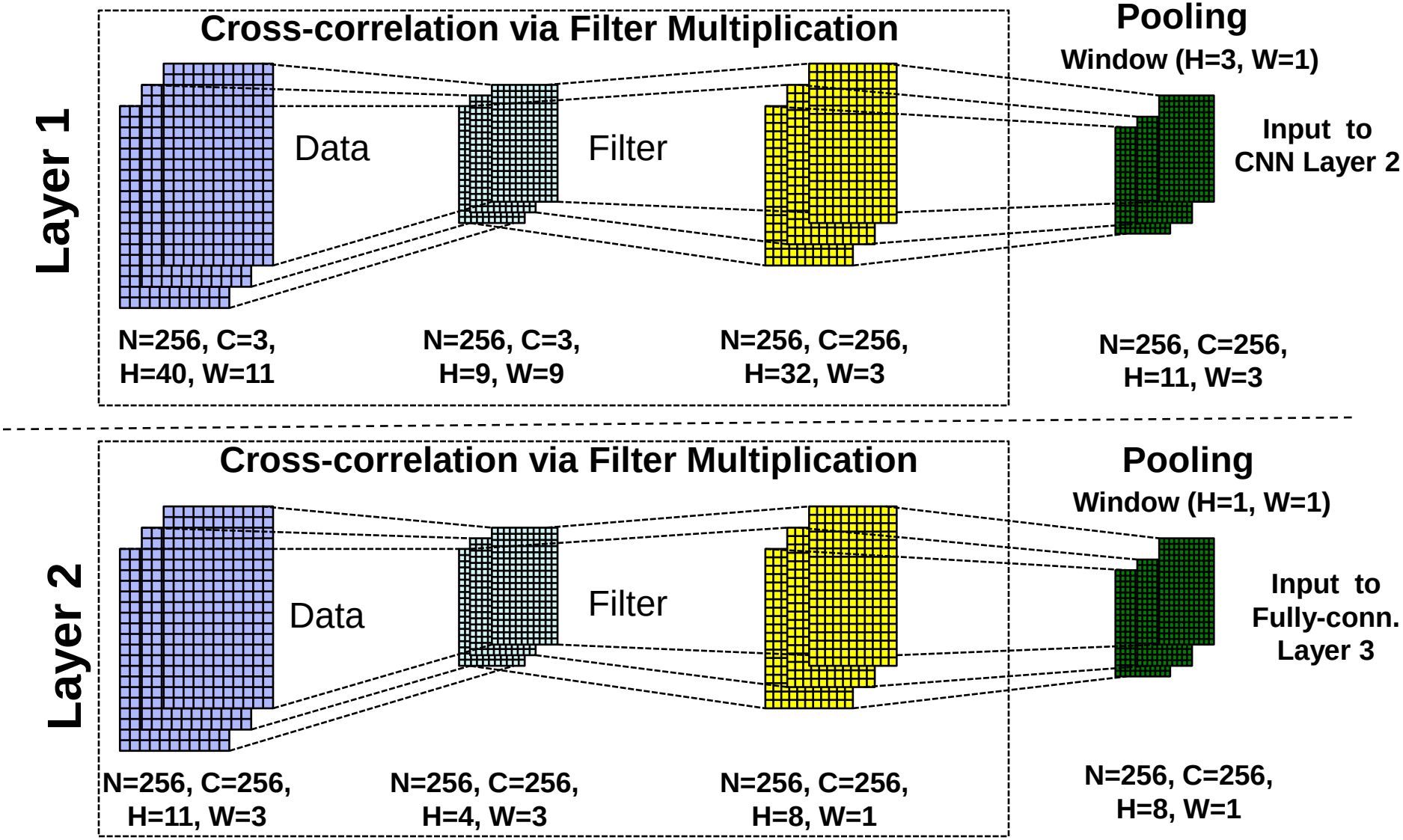
- Input signal sampled to 40*11 log-mel feature maps
- Input to the joint CNN/DNN system has mini-batches of 256 samples of 3 feature maps each: (N=256, C=3, H=40, W=11)
- An hour of speech contains around 1500 mini-batches

IBM Speech Processing System: Training Data Flow and Performance



- CNN costs dominate the end-to-end performance of current implementation.
- Each CNN stage requires around $3*10^9$ FLOPs per iteration per mini-batch. Usually, it takes 30 iterations for convergence.
- Forward pass uses cross-correlation, backward pass uses convolution
- **Current focus on optimizing CNN stages without pooling**

Convolution Neural Network Layers: Forward Operations

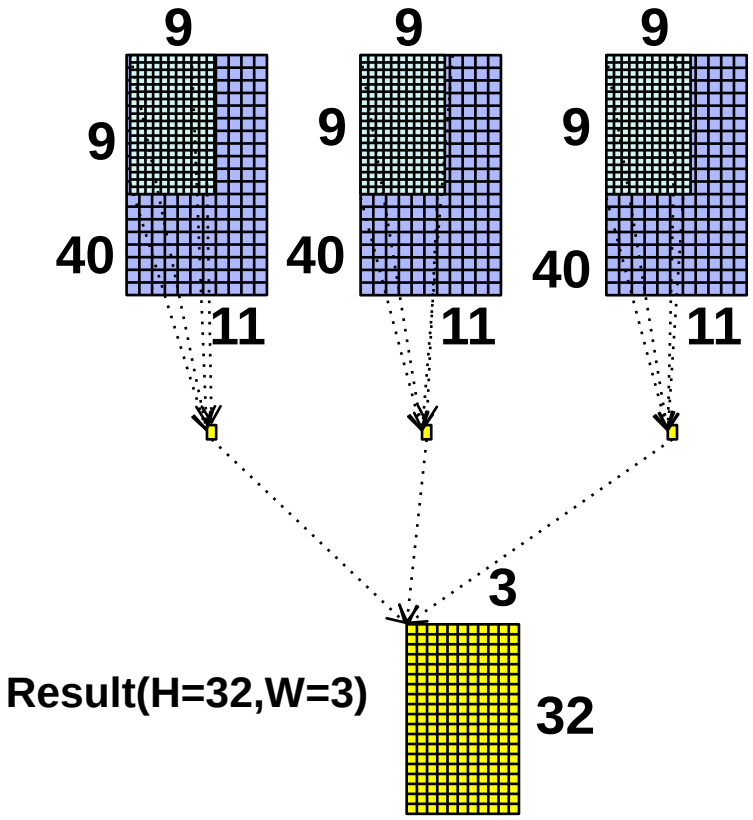


Convolution Neural Network Operations: Details

Cross-correlation

Data: (H=40, W=11, C=3)

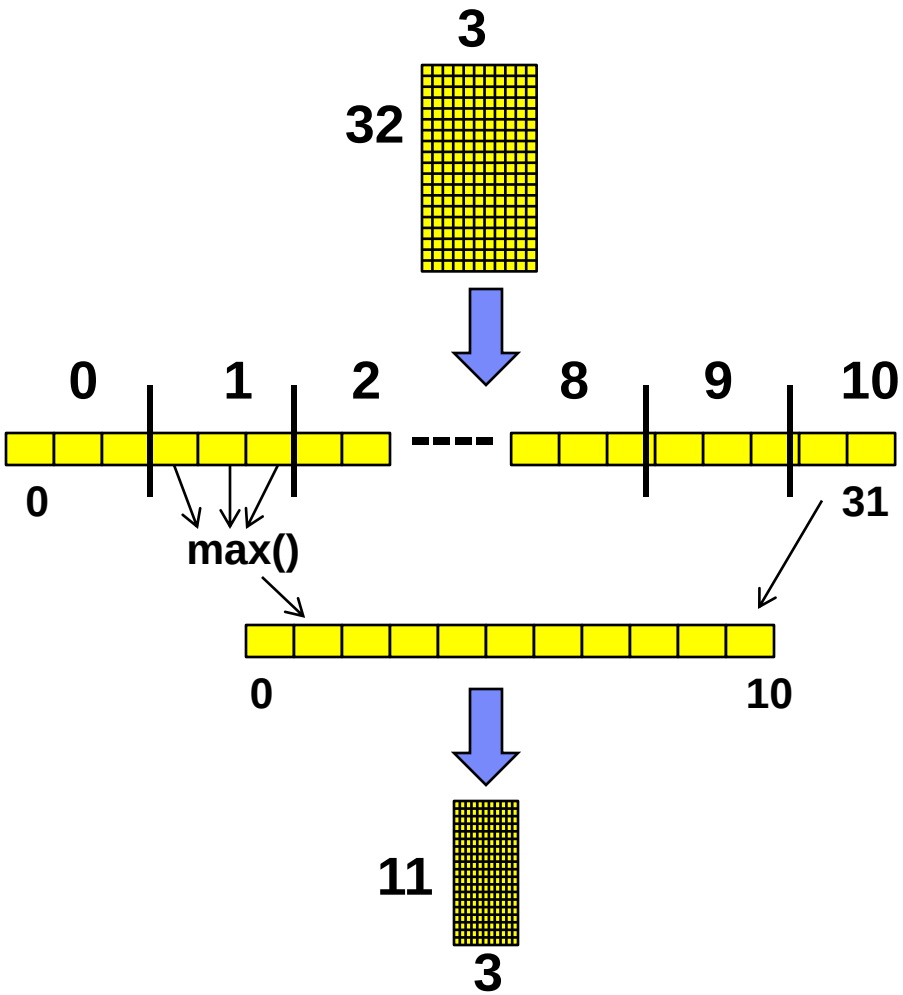
Filter: (H=9,W=9,C=3)



computing result(0,0)

Pooling window (H=3,W=1)

Input: (H=32,W=13) Output: (H=11,W=3)



Implementation of Convolution Neural Network Operations

- Three approaches for implementing correlation and cross-correlation operations on input and filter data

1. Matrix-based
   - Input and filter data expanded into matrices
   - Cross-correlation and Convolution implemented as a matrix-matrix multiplication (SGEMM)
   - Currently used by the IBM system (native) and internally by CUDNN
2. FFT-based
   - 3 Phases: FFTs of input and filter, frequency-domain dot-product, and inverse FFT
3. Custom implementations
   - Operations performed using existing shapes of the data and filter matrices

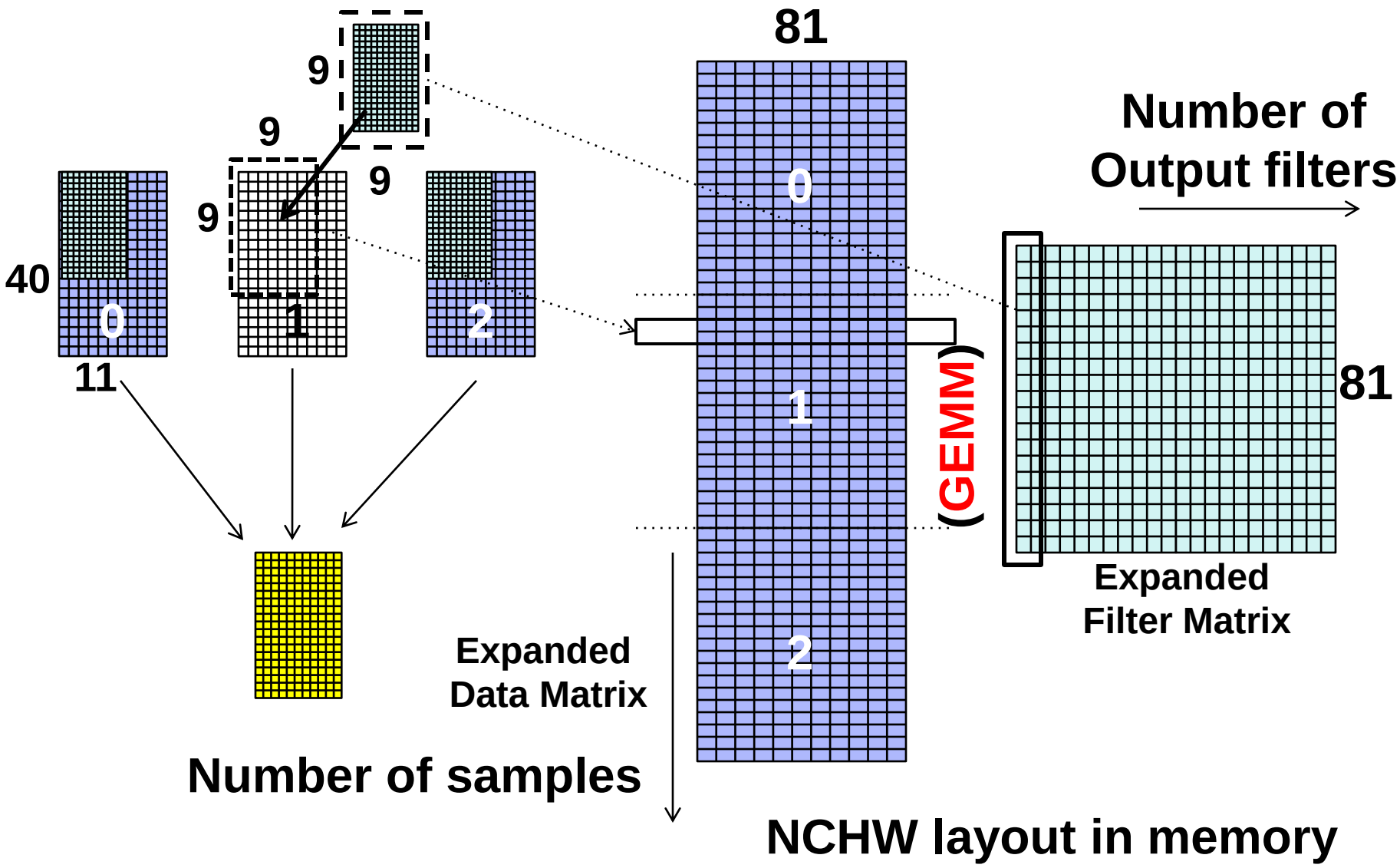
Matrix implementation of Cross-correlation (cc)

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \text{cc} \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} a*x+b*y+d*z+e*w & b*x+c*y+e*z+f*w \\ d*x+e*y+g*z+h*w & e*x+h*z+f*y+i*w \end{bmatrix}$$

expansion

$$\underbrace{\begin{bmatrix} a & b & d & e \\ b & c & e & f \\ d & e & g & h \\ e & f & h & i \end{bmatrix}}_{\text{Expanded Data Matrix}} * \underbrace{\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}}_{\text{Expanded Filter Matrix}} = \begin{bmatrix} a*x+b*y+d*z+e*w \\ b*x+c*y+e*z+f*w \\ d*x+e*y+g*z+h*w \\ e*x+h*z+f*y+i*w \end{bmatrix}$$

Matrix implementation of CNN Cross-correlation



Implementing Convolution (cv) using Cross-correlation (cc)

convolution

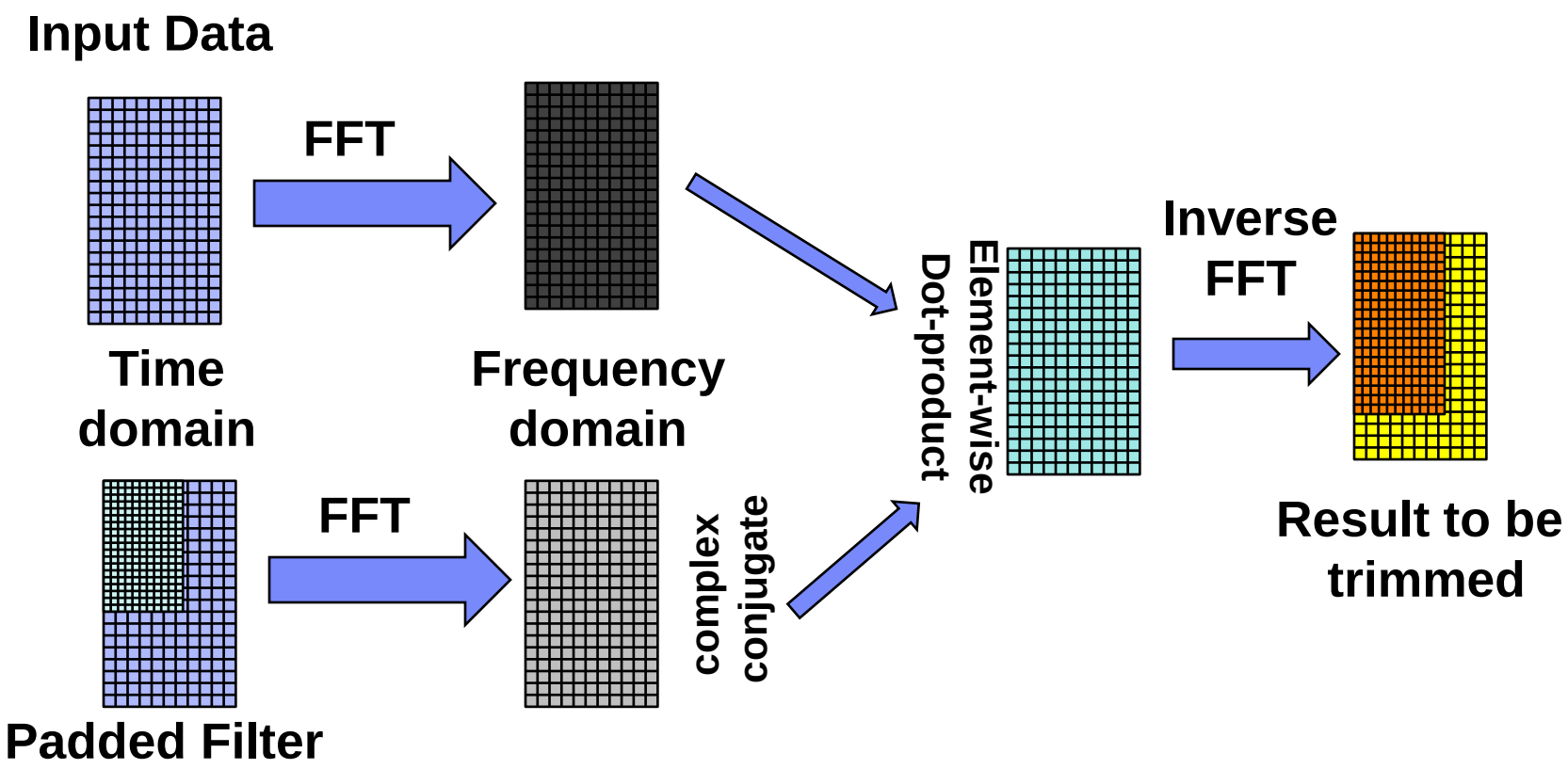
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \text{cv} \begin{bmatrix} x & y \\ z & w \end{bmatrix}$$

Convolution via cross-correlation

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & d & e & f & 0 \\ 0 & g & h & i & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{cc} \begin{bmatrix} w & z \\ y & x \end{bmatrix}$$

$$\begin{bmatrix} a*x & a*y+b*x & b*y+c*x & c*y \\ a*z+d*x & a*w+b*z+d*y+e*x & b*w+c*z+e*y+f*x & c*w+f*y \\ g*x+d*z & d*w+c*z+g*y+h*x & e*w+f*z+h*y+i*x & i*y+f*w \\ g*z & g*w+h*z & h*w+i*z & i*w \end{bmatrix}$$

Using FFTs for CNN Cross-correlation



Padding required to make sure input and filter sizes are the same.
Padding is also needed to prevent data wrapping around borders
Result needs to be trimmed

Custom Implementations (in progress)

- Custom implementation addresses problems with the matrix approach, e.g., space wastage, expansion..
- Custom implementation operates on the data as is
  - uses pair-wise multiplication and sum reduction of products to compute cross-correlation and convolutions
  - Makes use of warp functions and shared memory
- Current Issues
  - Filter and data sizes & shapes impact the performance

Experimental Setup

- Experiments run on a K40 using CUDA 6.5, and cuDNN v2 rc3 (Feb 27, 2015)
- Stand-alone CNN implementation of the two layers, with the exact parameters used in the production system
- Evaluated forward, backward propagation (data gradient) and gradient (filter gradient) computations using native and cuDNN approaches

Performance of CNN Layer 1: native & cuDNN

Method	Forward	Data Gradient	Filter Gradient
cuDNN v2-rc3 Implicit GEMM	2.13	2.49	2.84
cuDNN v2-rc3 Pre-comp GEMM	**1.60**	**2.50**	**2.84**
cuDNN v2-rc3 GEMM	2.11	2.49	2.84
cuDNN v2-rc3 DIRECT	------	2.49	2.84
cuDNN v2-rc3 Gradient using Forward Operator	**1.61**	**9.45**	**7.28**
IBM Native	**1.79**	**2.09**	**3.42**

Time in ms. Output in NCHW format.

Performance of CNN Layer 2: native & cuDNN

Method	Forward	Data Gradient	Filter Gradient
cuDNN v2-rc3 Implicit GEMM	3.17	19.88	22.98
cuDNN v2-rc3 Pre-comp GEMM	**2.02**	**19.88**	**22.87**
cuDNN v2-rc3 GEMM	2.19	19.90	22.87
cuDNN v2-rc3 DIRECT	------	19.90	22.87
cuDNN v2-rc3 Gradient using Forward Operator	**2.05**	**4.77**	**1.99**
IBM Native	**2.16**	**2.38**	**1.99**

Time in ms. Output in NCHW format.

CNN Performance for Speech Processing

- Speech processing CNN computation characterized by small data and filter blocks with irregular shapes
- Overall cuDNN performance does not present an advantage for our application over the native approach.
  - cuDNN forward propagation using pre-computed indices performs better than the native approach
  - cuDNN backward and gradient computations on second layer perform poorly. Performance can be improved by invoking cuDNN forward computation on padded data.
  - cuDNN behavior sensitive to the data layout
- Current versions of custom implementations slower
- Current FFT-based techniques slower than cuDNN for the layer 1 and competitive for Layer 2

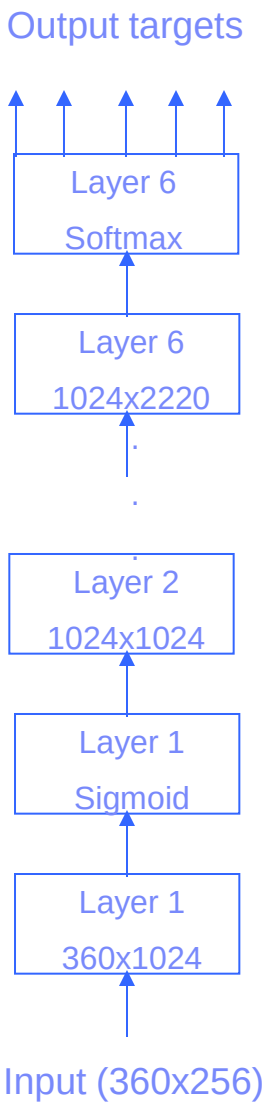
Conclusions and Future Work

- GPUs are very effective in accelerating CNN computations in speech processing.
- Speech processing computation characterized by small and irregularly shaped data samples
- cuDNN performance degrades in certain conditions/operations (e.g., backward and gradient)
- For the current configuration, use of FFTs for cross-correlation/convolution is not effective

BACKUP SLIDES GO HERE

Deep Neural Networks Details

- A feed-forward DNN has multiple hidden fully-connected layers, where each layer consists of
  - a **linear weight matrix**
  - a **non-linear activation function (e.g., Softmax)**
- Outputs targets:
  - Number of classes (i.e., sub-word units, words)
  - Output probabilities used as scores
- Multiple layers make DNNs a highly non-convex optimization problem
- Most popular approach for training is mini-batch Stochastic Gradient Descent (SGD)

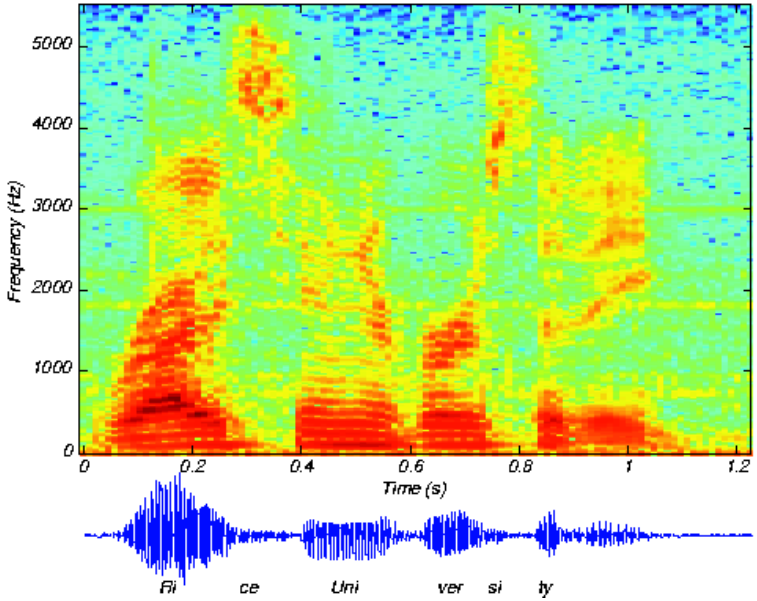


Convolutional Neural Networks (CNNs)

- What DNNs do not model well:

1. Input correlations
2. Translation variance

- CNNs are more powerful than DNNs because they **reduce spectral variations and model correlations** in the input signal, **jointly while performing discrimination**
- With fewer parameters, CNNs provide better accuracy than DNNs



[Sainath et al, ICASSP 2013]

FFT and CNNs

- **Cross-correlation of data blocks in mini-batches and filters can be performed in "frequency" domain by computing element-wise products between their Fourier transforms.**

- **The DFT requires data blocks and filters to have the same size, so padding filters and data (to prevent data wrapping on the borders due to the circular convolution for backward pass) is needed.**

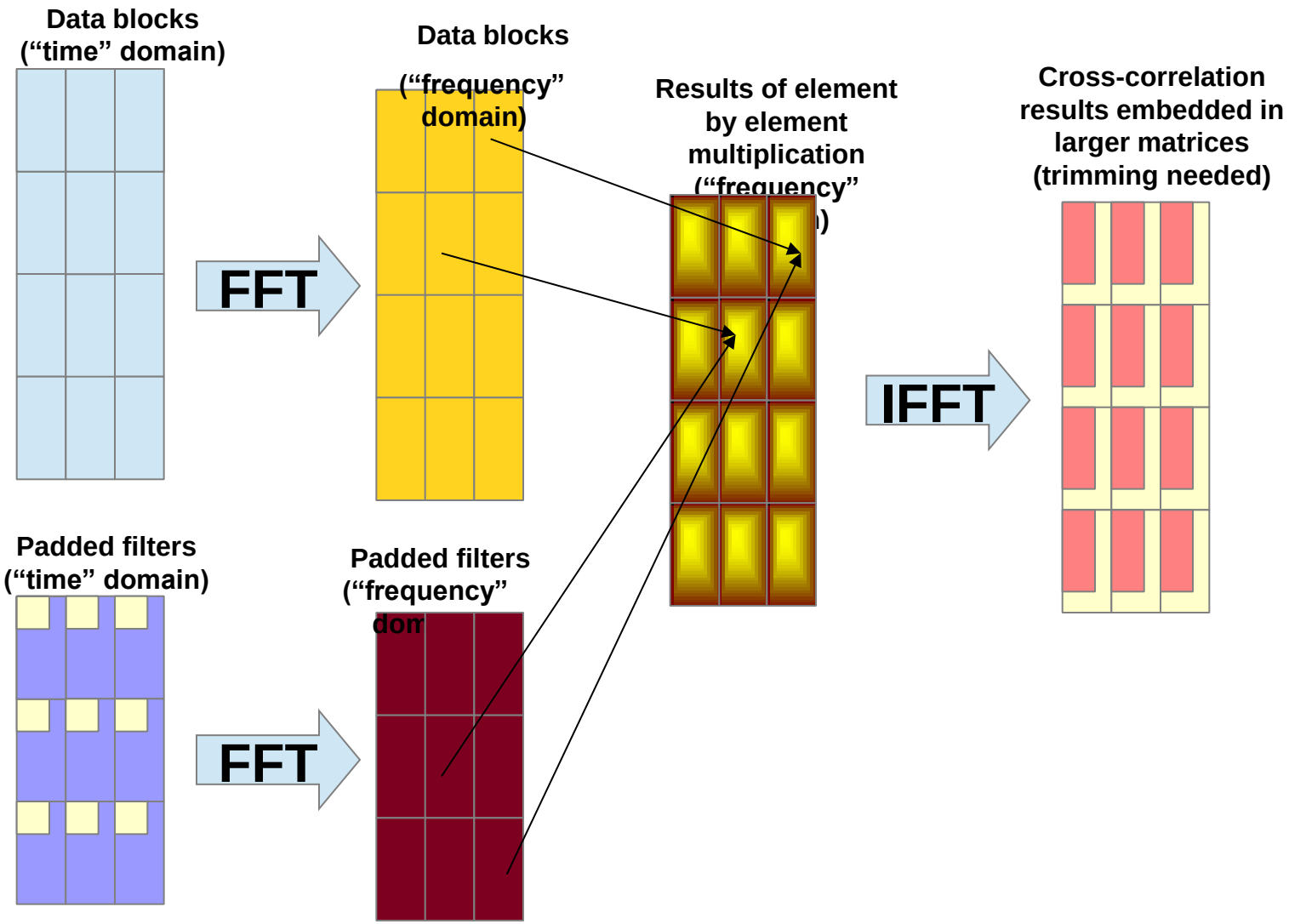
- **Filters must be either flipped in "time" domain before the FFT or be complex conjugated in "frequency" domain to ensure the cross-correlation is computed correctly.**

- **Once the element-wise product of the data and filter transforms is computed, it is transformed back to "time" domain by inverse FFT.**

- **The result in time domain is a batch of padded convolved matrices. Therefore, trimming is necessary to get the results identical to those computed by linear convolution.**

- **Implementation details: the batch version of CUFFT was used, several kernels were written to pad/trim, data/filters, multiply transform element by element, etc.**

FFT and CNNs (work flow)



Exploiting FFTs for CNN Cross-correlation

- Cross-correlation in frequency domain by computing element-wise products between the Fourier transforms of input and filter data
  - Filter data needs to be complex conjugated before FFT
  - Inverse FFT on the element-wise products to get results in time domain
- Padding needed during FFT (trimming the results)
  - DFT requires input and filter data to be of the same size
  - Prevents data wrapping on the borders