



CEVA[®]



Deep Neural Networks in Embedded and Real Time Systems

www.ceva-dsp.com

Deep Learning Neural Networks



► Deep Learning

► A family of neural network methods using high number of layers

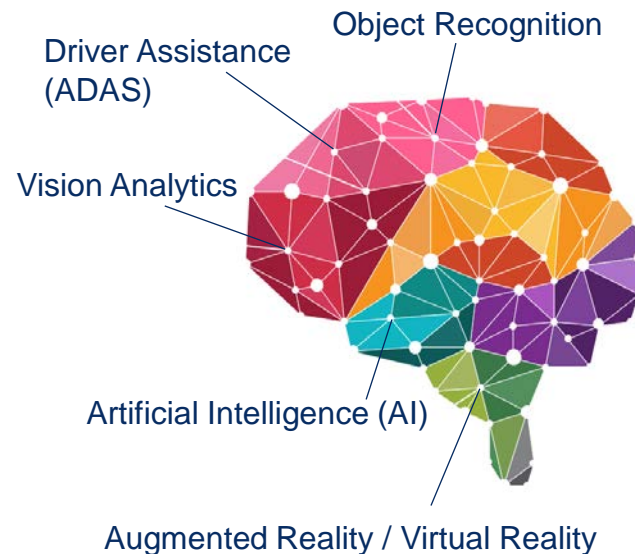
► Focused on feature representations

► Convolutional Neural Networks (CNN)

► Most popular deep learning neural network method

► Benefits

1. Best recognition quality (vs. alternative recognition algorithms)
2. Re-trainable without code changes (implemented once and used many times)



Training is for everyone



- ▶ Free available training frameworks:
 - ▶ **Caffe** – The most popular, developed by the Berkeley Vision and Learning Center
 - ▶ **TensorFlow** – Open source software library for numerical computation using data flow graphs supported by Google
- ▶ Training is done offline
- ▶ Training and designing a network became an expertise
- ▶ There are many ways to design a network and train it
- ▶ Large variety in network structures (GoogLeNet, AlexNet, VGG, NIN)
- ▶ Many open source examples are available (Caffe Model Zoo)



RT classification requires knowledge



- ▶ There are many ways to design a CNN
 - ▶ Optimizations are usually done to a specific network
 - ▶ There is a strong requirement for flexible implementations
 - ▶ Modifying the network
 - ▶ Changing network characteristics
- ▶ Porting proprietary networks consumes time
 - ▶ Special programming knowledge (intrinsic, assembly)
 - ▶ Specific experience in the embedded platform (instructions, hardware capabilities)
 - ▶ Optimized solutions are not flexible for changes



Long “Time To Market”

Neural network embedded challenges



CEVA®

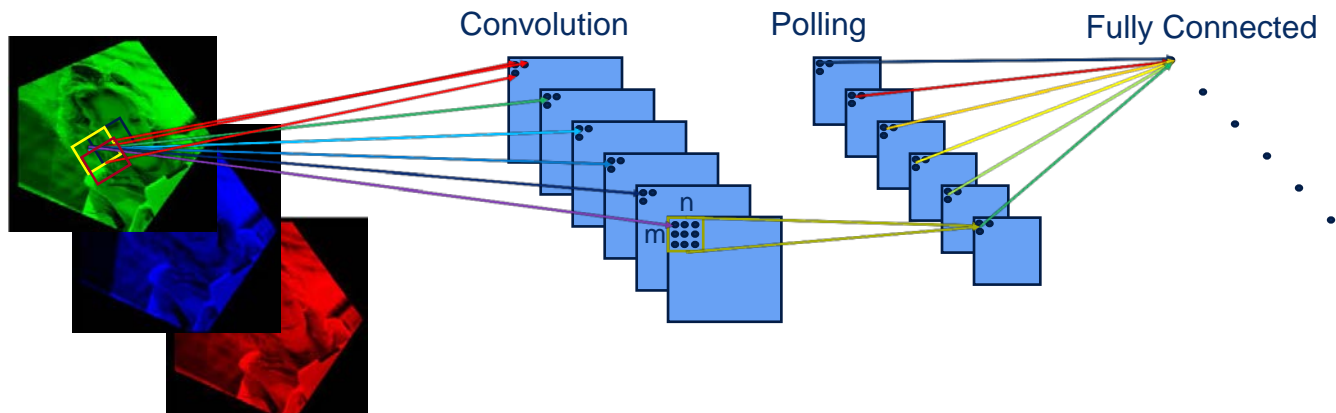
- ▶ Very high bandwidth consuming
 - ▶ Between Layers of data transfer in and out the DDR
 - ▶ AlexNet – 12 MB between layers data transfer (16-bit precision single execution)
 - ▶ Convolution and Fully Connected data weights from DDR
 - ▶ AlexNet – 243 MB weights in floating point precision
 - ▶ Processing more than one ROI with the same network
- ▶ Required conversion from floating point to fixed point
 - ▶ Training results are in floating point while low power and area platforms prefer fixed point calculations to reduce power and area
- ▶ The number of operations can reach more than a mega operations per layer especially in the convolution layer
- ▶ Internal memory size limitation on embedded platform



What can be done? (1)

▶ Reducing bandwidth

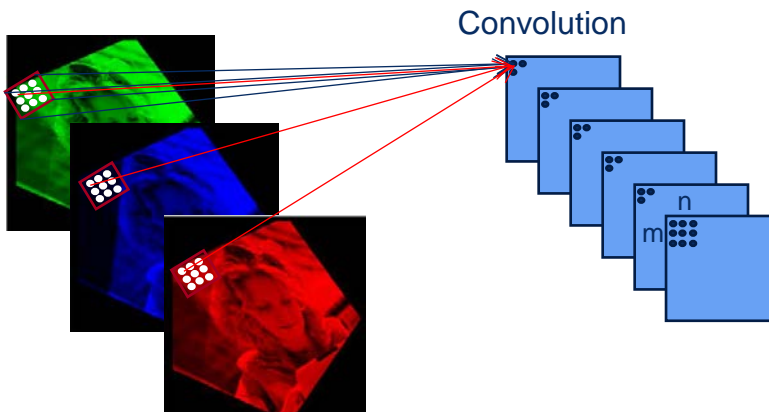
- ▶ In the convolution layer, each output is calculated by the same inputs
 - ▶ Weights matrix are shared between output results in the same map (in order not to load the weights more than once)
 - ▶ The input data can be reused to avoid useless transactions from DDR



What can be done? (2)

▶ Maximum multiply accumulate utilization

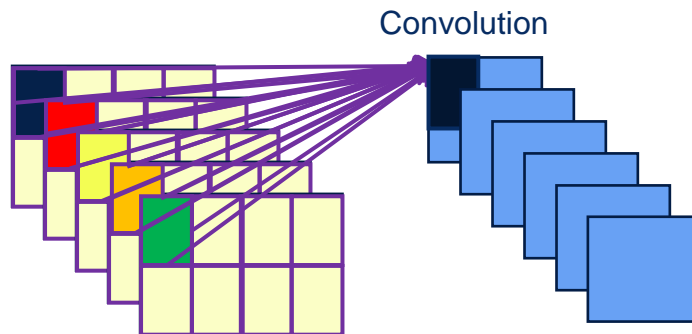
- ▶ Differentiating between large inputs to small input maps and the amount from each type
- ▶ Large size maps - $X_{c,i,j}^l = \sum_{c=0}^C \sum_{i=0}^H \sum_{j=0}^W W_{c,m,n}^l X_{c,i+m,j+n}^{l-1}$
- ▶ Many maps with small size (last layers) - $X_{c,i,j}^l = \sum_{i=0}^H \sum_{j=0}^W \sum_{c=0}^C W_{c,m,n}^l X_{c,i+m,j+n}^{l-1}$



What can be done? (3)

► Overcome small internal memory size

- Trying to preserve the principle of “All inputs must be in the internal memory” by tile division
- Dividing all input maps to same tile size



What can be done? (3)



- ▶ Using compression algorithms and prior knowledge to reduce bandwidth to and from the external memory
 - ▶ It is known there is a lot of redundancy in the network data
 - ▶ Can be done offline
 - ▶ Example
 - ▶ AlexNet fully connected BW, before compression, can be reduced to 6 MBytes



- ▶ Using dedicated and smart instructions for large scale operations such as convolutions
 - ▶ Example: CEVA vector processing by CEVA-XM4 DSP core
 - ▶ Includes dedicated instructions which help with CNN accelerator



CEVA Deep Neural Network Library



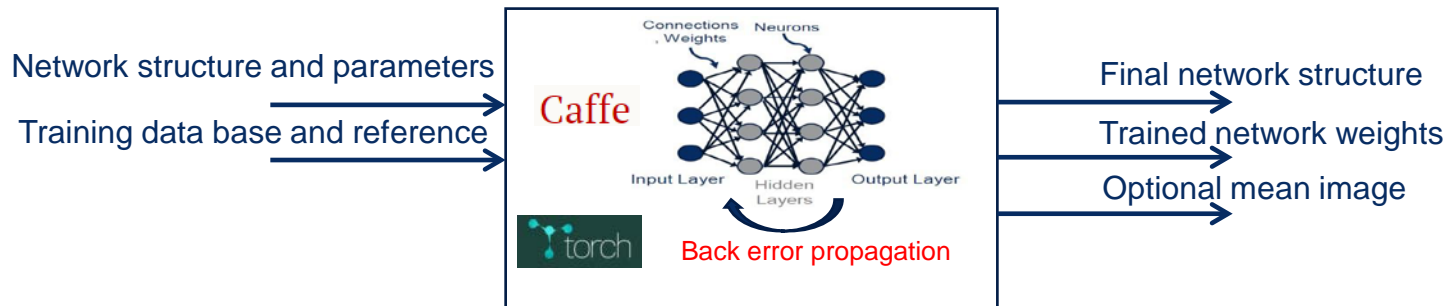
- ▶ General acceleration library for deep neural network algorithms especially convolution neural network (CNN)
- ▶ Provides offline tool for network generator
 - ▶ Converts from offline to real-time network representation
 - ▶ Converts floating point trained network weights to fixed point weights with small degradation
 - ▶ Optimizes the initial network structure to real time execution
- ▶ Real-time initialization and execution of any designed network
- ▶ Supports single layer acceleration such as:
 - ▶ Convolution, pooling, softMax, normalization, activation and fully connected

Programmer Flow for CNN Acceleration (1)



Offline training

- ▶ Network developer uses proprietary training process
 - ▶ In-house framework
 - ▶ Open source frameworks
- ▶ Training output :
 - ▶ Network weights in floating point precision
 - ▶ Final network defined structure



Programmer Flow for CNN Acceleration (2)



CEVA Network Generator

- ▶ Converts weights and network structure definition to real-time execution on CEVA-XM4
 - ▶ Floating point to fixed point network weights
 - ▶ Network optimizations



Programmer Flow for Fast CNN Acceleration (1)



Programmer interface : Network creation and initialization (done once)

► CDNN context creation

```
/* create and initialize the CEVA deep neural network context */
status = CDNNCreate(pCDNNHandle);
```

► Memory buffers description (output and input)

```
cdnn_databuffer_parameters_t imageToCDNNstructIn;
imageToCDNNstructIn.nInputs = 1;
imageToCDNNstructIn.width = 400;
imageToCDNNstructIn.height = 400;
imageToCDNNstructIn.nChannels = 1;
imageToCDNNstructIn.dataOrder = E_CDNN_MEMORY_DATAORDER_NHWC;
imageToCDNNstructIn.depth = E_CDNN_PRECISION_16BIT;
imageToCDNNstructIn.dataType = E_CDNN_DATATYPE_S16;
cdnn_datab inputImage = CDNNCreateDataBuffer(*pCDNNHandle, &imageToCDNNstructIn);
```

► Network creation

```
intNetworkParams_st networkParam;
networkParam.pInputBuffer = inputImage;
networkParam.pOutputBuffer = *outputImage;
networkParam.outputLayerId = 11;
networkParam.networkMode = E_CDNN_NETWORK_MODE_SLIDINGWINDOW;
networkParam.pNetwork = pNetworkFilename;
cdnn_network network = CDNNCreateNetworkFromFile(*pCDNNHandle, &networkParam);
```

► Network initialization

```
/* init CDNN context */
status |= CDNNInitialize(*pCDNNHandle);
```

Programmer Flow for Fast CNN Acceleration (2).



CEVA®

Programmer interface : Network execution (streaming)

► Update network input memory buffer

```
cdnn_datab inputImage = CDNNCreateDataBufferFromHandle(pCDNNHandle, &imageToCDNNStructIn, inImg.data);  
s32status |= CDNNNetworkUpdateParameter(network, (cdnn_reference)inputImage, 0);
```

► Execution

```
/* classify the image */  
s32status |= CDNNNetworkClassify(pCDNNHandle, network);
```

► Query for results

```
s32status |= CDNNQueryDataBuffer(pOutputLayer, E_CDNN_BUFFER_ATTRIBUTE_WIDTH, &width, sizeof(width));  
s32status |= CDNNQueryDataBuffer(pOutputLayer, E_CDNN_BUFFER_ATTRIBUTE_HEIGHT, &height, sizeof(height));  
s32status |= CDNNQueryDataBuffer(pOutputLayer, E_CDNN_BUFFER_ATTRIBUTE_INPUTS, &inputNumber, sizeof(inputNumber));  
s32status |= CDNNQueryDataBuffer(pOutputLayer, E_CDNN_BUFFER_ATTRIBUTE_CHANNELS, &channels, sizeof(channels));  
s32status |= CDNNAccessDataBuffer(pOutputLayer, &pOutData);
```

► Update OpenCV data structures

```
Mat resultMat(channels, inputNumber, CV_64F, pOutData);
```

Programmer Flow for Fast CNN Acceleration (3).



CEVA®

Programmer interface : Network release

- ▶ Release all open CDNN memory object created

```
status |= CDNNReleaseDataBuffer(pCDNNHandle, &outputImage);
```

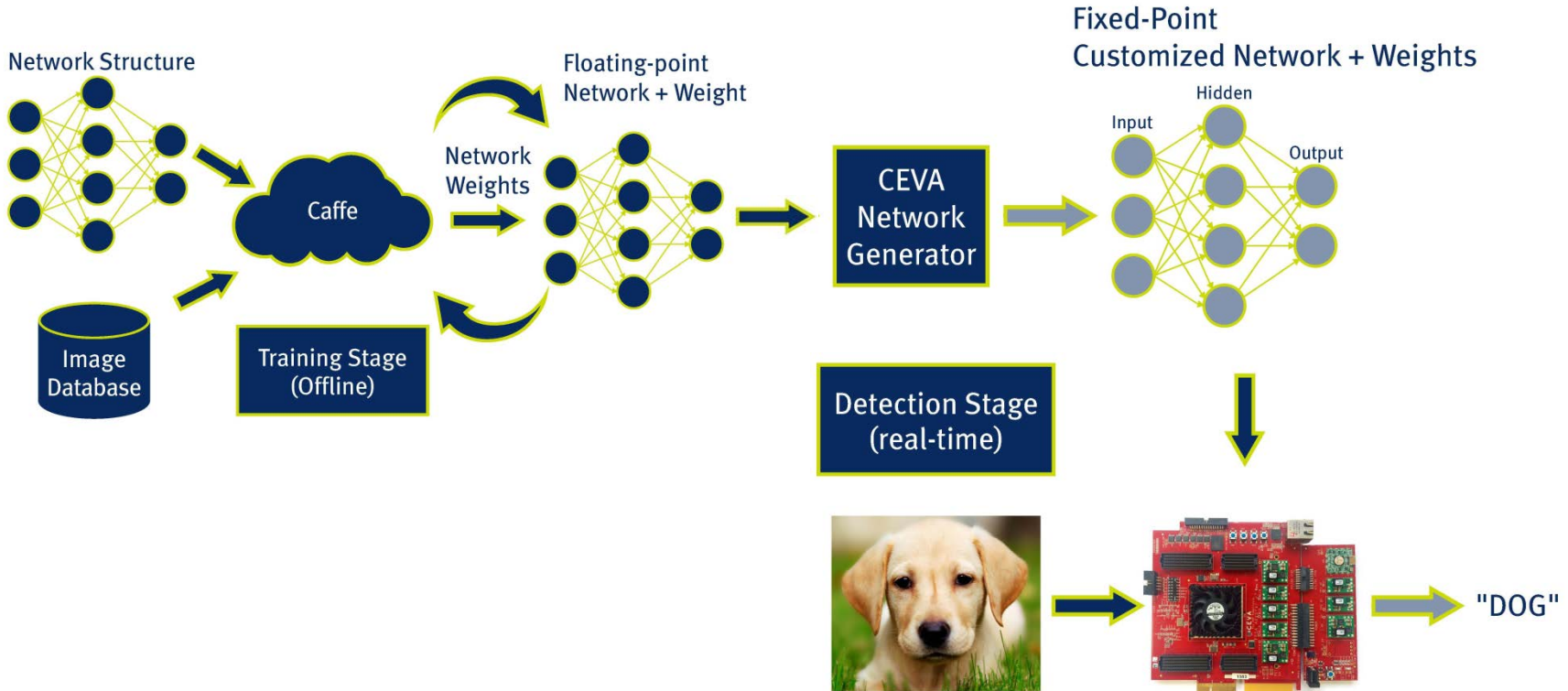
- ▶ Release the CDNN Network

```
status |= CDNNReleaseNetwork(pCDNNHandle, &network);
```

- ▶ Release CDNN

```
status |= CDNNRelease(pCDNNHandle);
```

CNN Usage Flow with Caffe & CDNN





Thank You

www.ceva-dsp.com