Deep Learning with Intel DAAL on Knights Landing Processor

David Ojika

dave.n.ojika@cern.ch

March 22, 2017

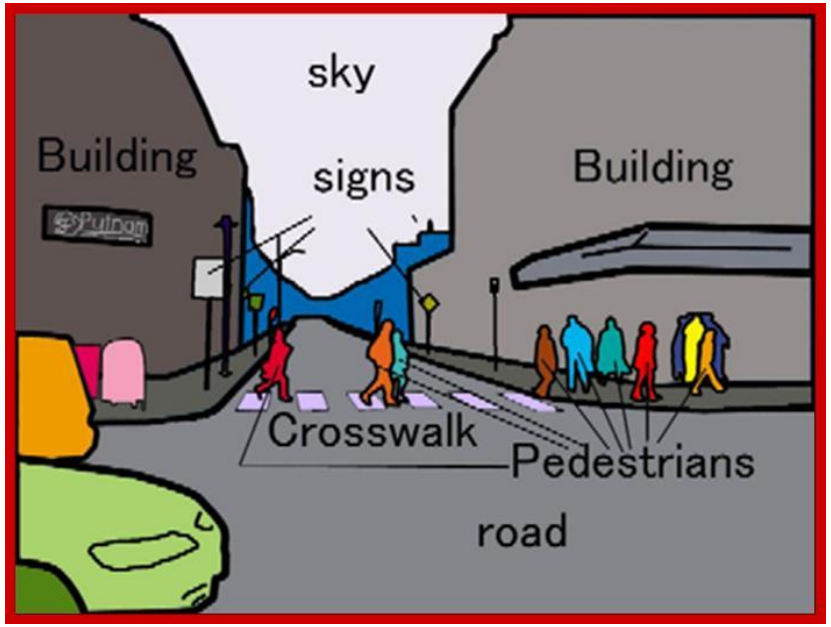
Outline

- Introduction and Motivation
- Intel Knights Landing Processor
- Intel Data Analytics and Acceleration Library (DAAL)
- Experiment and current progress

GitHub: https://github.com/davenso/DKN

Object Identification

- Object identification is the task of **training** a computer to **recognize** patterns in data
  - **Object**: could be car, person, a piece of flower, higgs, muons, etc.
  - **Learning algorithm** (e.g. decision tree) is used to for training

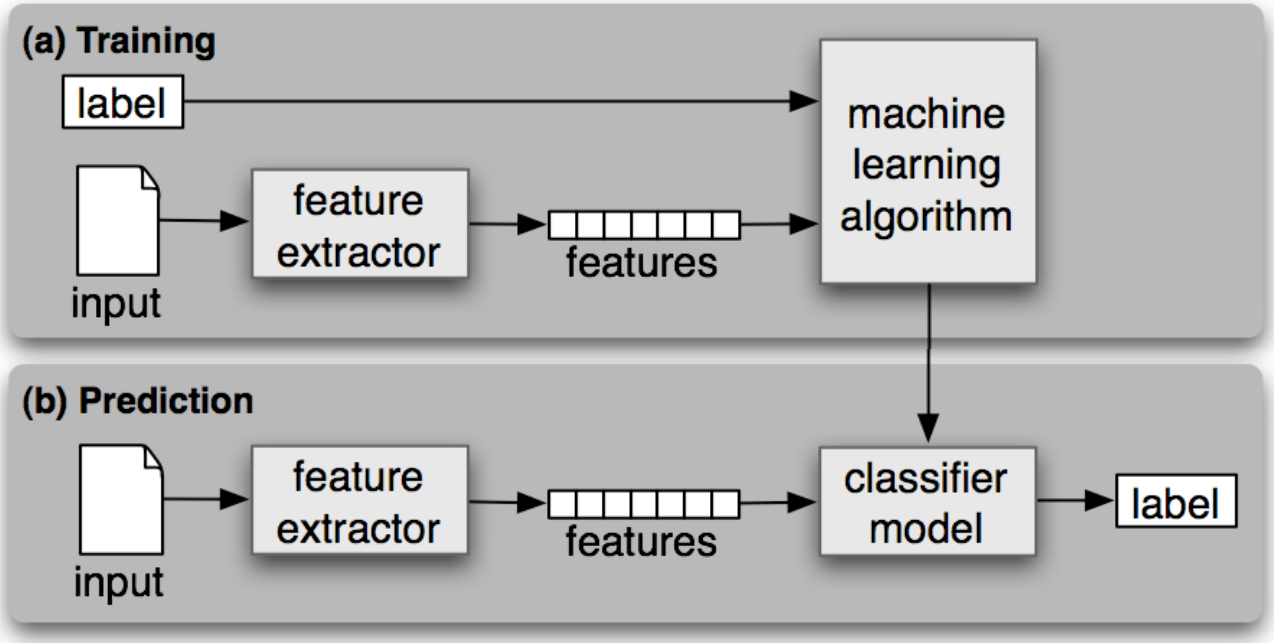


Generate a "prediction model" to make "guesses"

http://www.cs.tau.ac.il/~wolf/OR/

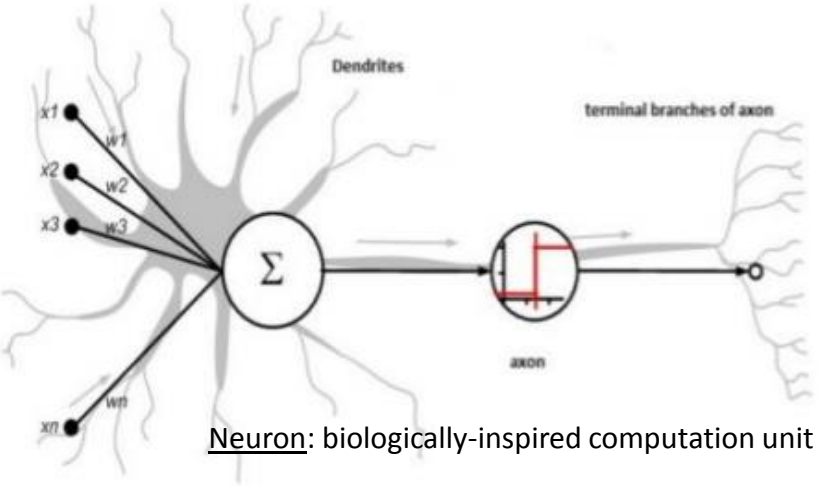
Machine Learning

- Involves two 2 steps:

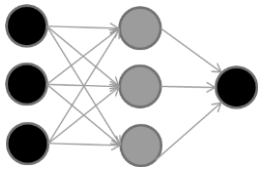


Deep Learning

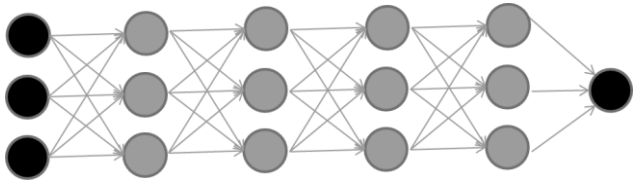
- Deep learning means using **neural networks** (a class of machine learning) with **multiple hidden layers**
  - Neural networks are modelled based on the **dynamics of the neurons** in our brain
  - *Hidden layers* represent neural computations in series of *processing stages*
  - Learning performance can generally improve with depth of network (at more cost to processing)



Neuron: biologically-inspired computation unit



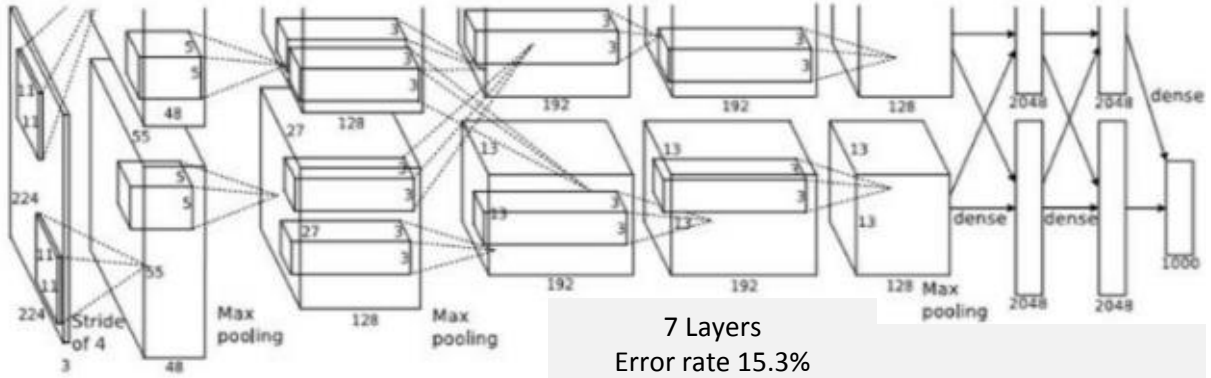
Neural Network



Deep Neural Network (DNN)

DNN Applications

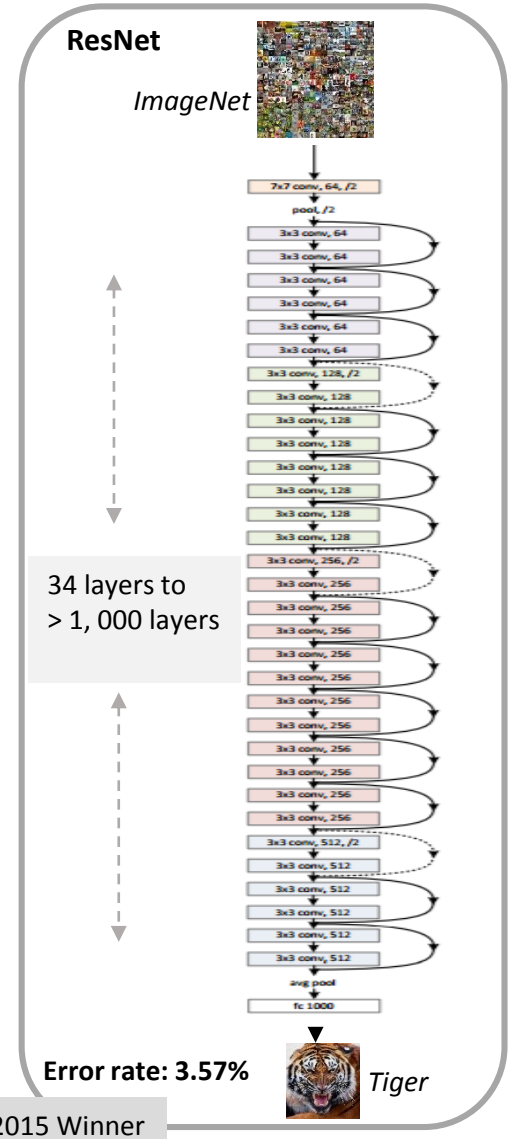
Handwriting



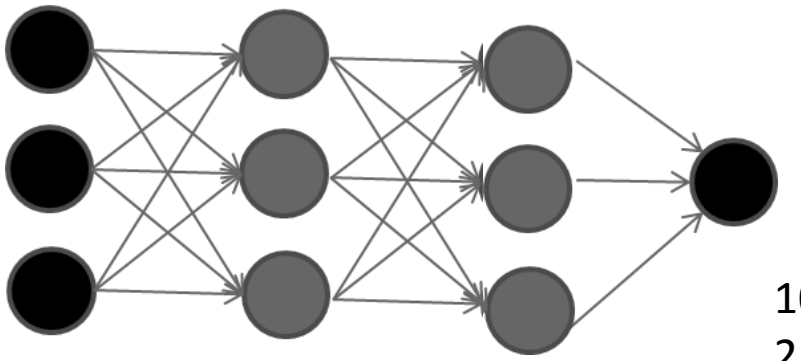
1 2 3
4 5 6
7 8 9
0

Images

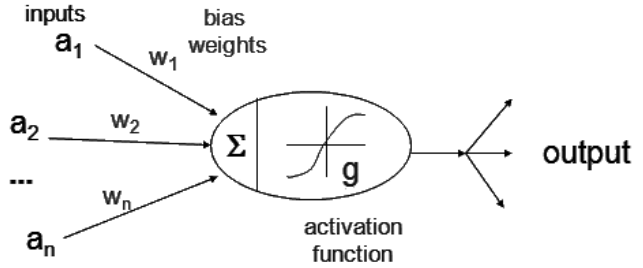
Tiger

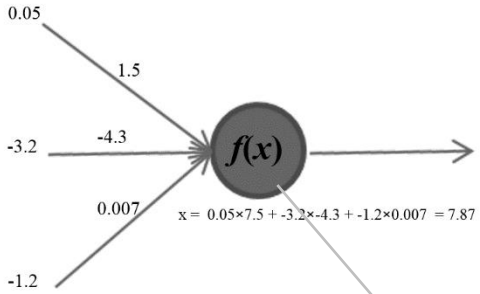


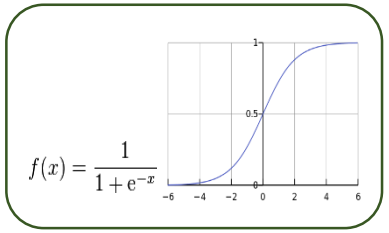
DNN Training



10 Neurons
2 Hidden Layers



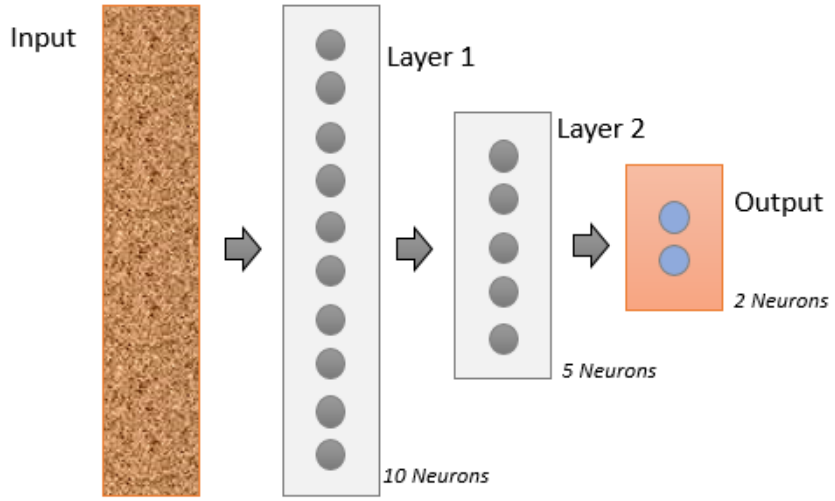




- Back propagation and forward propagation method used training

DNN Inferencing

A trained network with:
- 10 Neurons
- 2 Hidden Layers



Layer 1

0.009	-0.029	0.023	0.062	0.04	-0.065	-0.022	0.011	-0.037	0.014	-0.011	0.001	-0.036	0.034	-0.003	0.011	0.006	-0.003	0.016	-0.033
-0.005	0.015	-0.011	-0.032	-0.02	0.032	0.011	-0.006	0.019	-0.008	0.005	0	0.019	-0.016	0.003	-0.004	-0.004	0.002	-0.008	0.017
0.011	-0.032	0.026	0.068	0.044	-0.072	-0.024	0.012	-0.041	0.016	-0.013	0.002	-0.04	0.039	-0.004	0.011	0.006	-0.003	0.017	-0.037
-0.018	0.054	-0.043	-0.116	-0.078	0.123	0.041	-0.02	0.069	-0.026	0.022	-0.004	0.069	-0.065	0.008	-0.019	-0.011	0.005	-0.031	0.062
0.004	-0.014	0.011	0.03	0.019	-0.032	-0.011	0.006	-0.017	0.007	-0.005	0	-0.018	0.016	-0.001	0.005	0.004	-0.002	0.008	-0.017

Learned Weights (10 x 5)

0.114
0.223
0.222
0.497
0.184

Learned Biases (1 x 5)

Layer 2

0.613	0.721	0.721	0.993	0.683
0.787	0.635	0.915	0.669	0.769

Learned Weights (5 x 2)

0.618
0.72

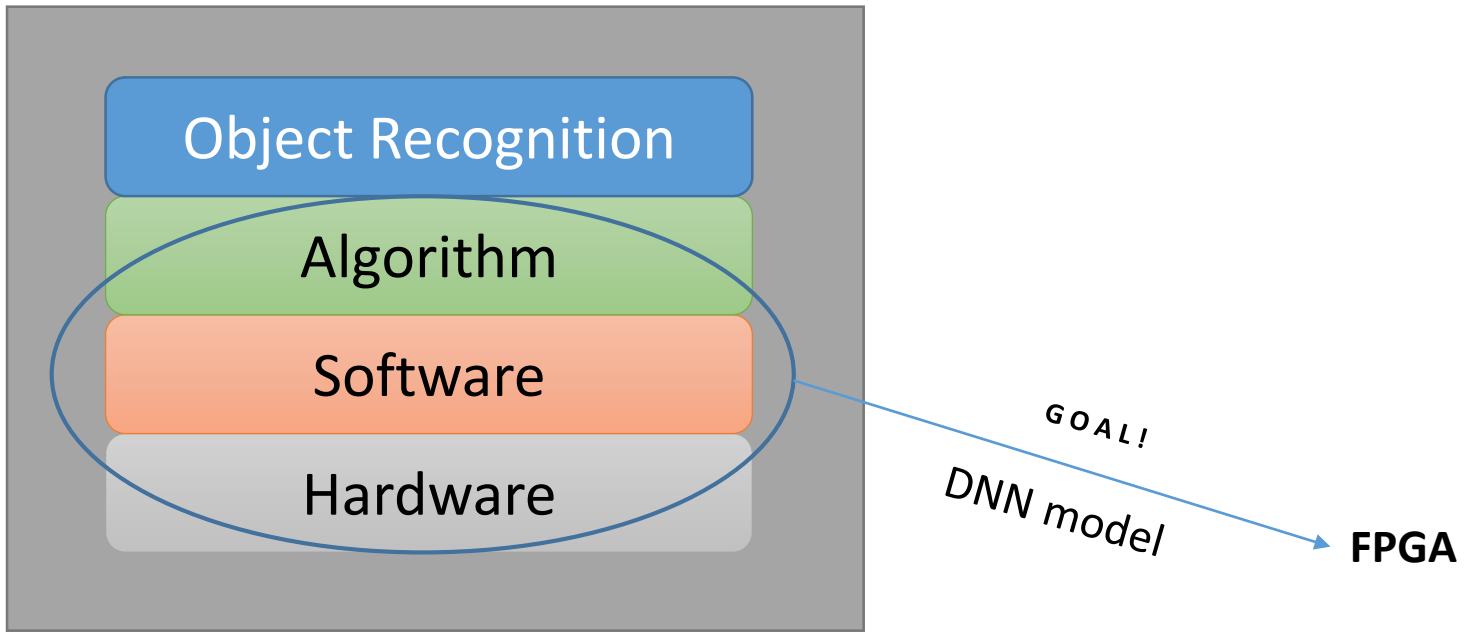
Learned Biases (1 x 2)

Challenge with DNN Training

- Large size of dataset
  - Gigabytes, Terabytes of data
- Large number of hyper-parameters
  - # layers, # neurons, batch size, iterations, learning rate, loss function, weight decay etc.
    - Hyperparatmer optimization techniques: random search, grid search, Bayesian, gradient-based
- Emerging hardware for deep learning
  - GPU
  - KNL
  - FPGA
- Software exists, but require manual fine-tuning

(Physics) Object identification: A Cross-layer Perspective

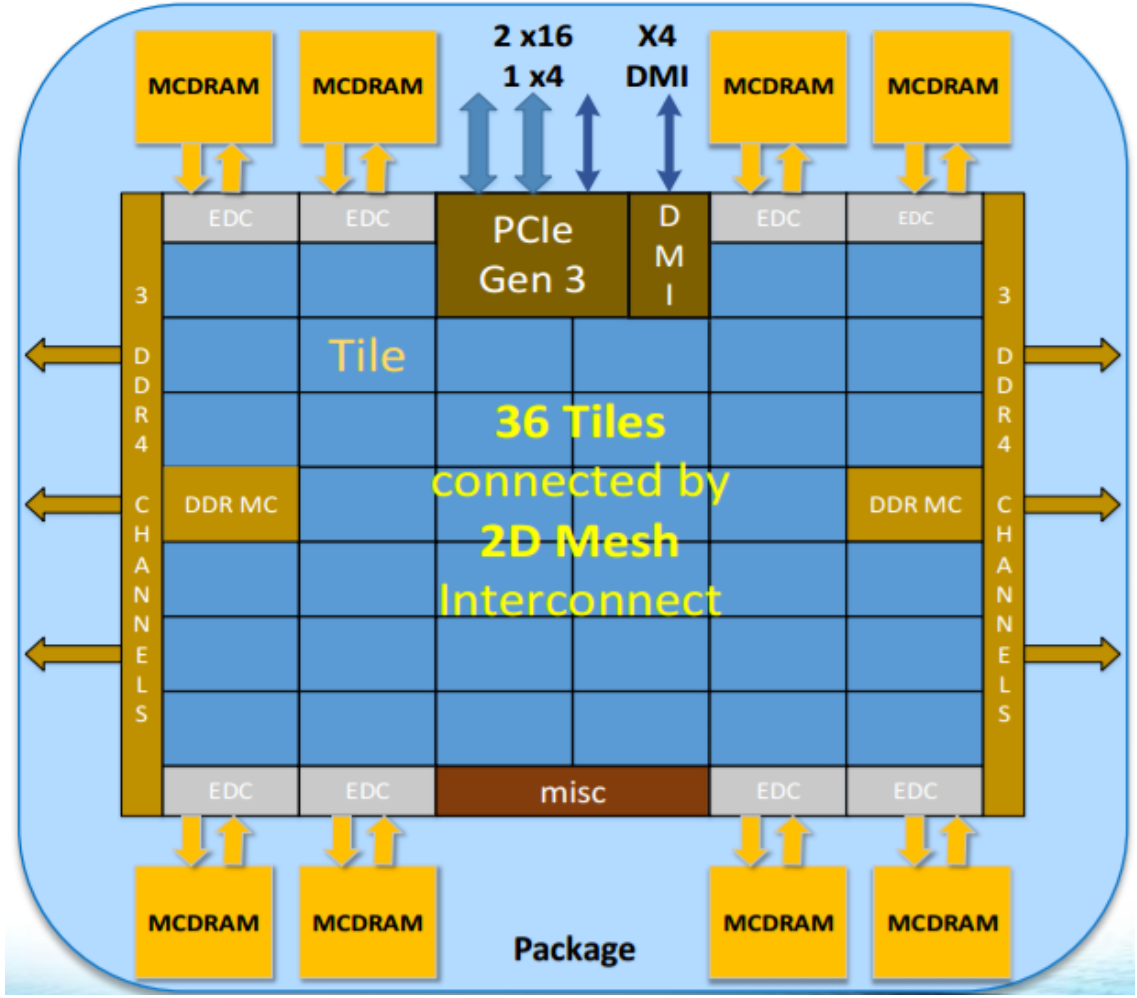
- Compose hardware, algorithm and software and components
- Derive efficient FPGA implementation to perform inference



Intel Knights Landing (KNL)

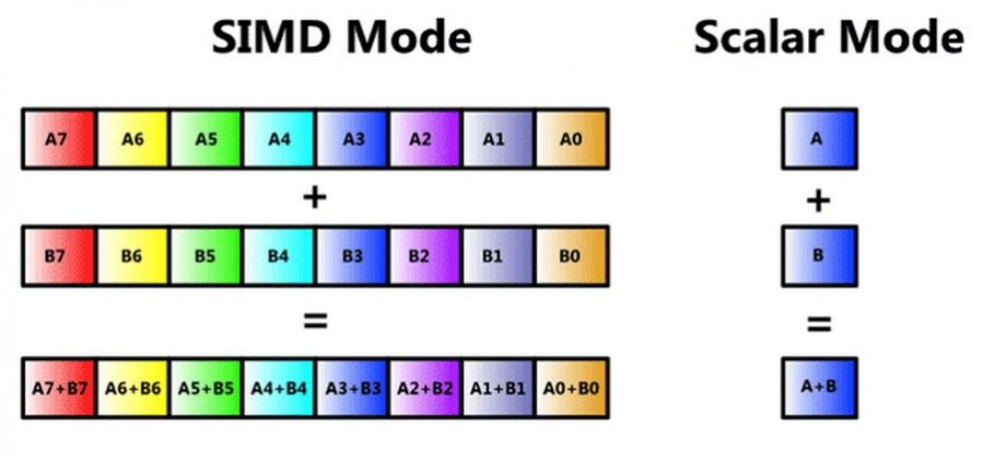
- Next generation Xeon Phi (after Knight Corner (KNC) co-processor)
  - **Self-boot:** unlike KNC
  - **Binary-compatible** with Intel Architecture (IA) and boots standard OS
- Some performance-enhancing features
  - Vector processors: AVX-512
  - High-bandwidth: MCDRAM
  - Cluster modes
  - Networking (in some models)

KNL Overview

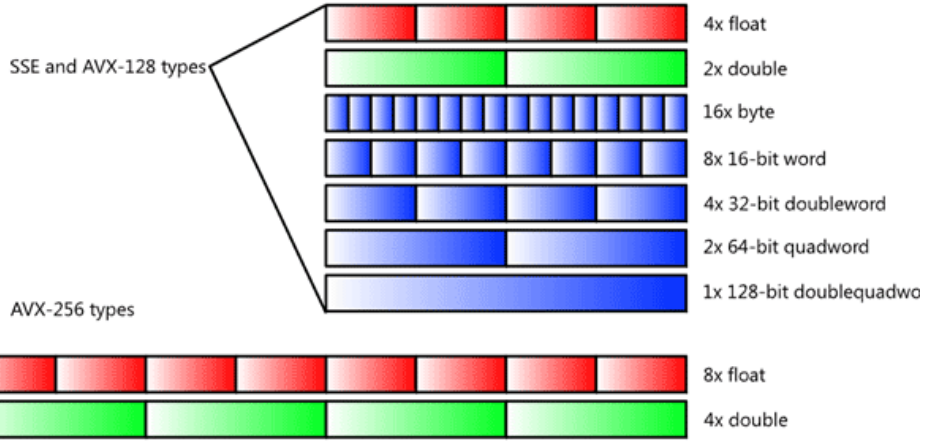


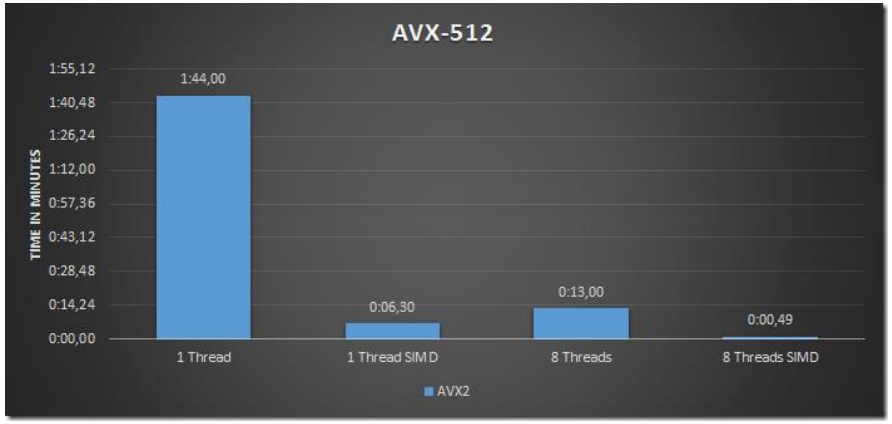
- Chip: 36 Tiles interconnected by 2D Mesh
- Tile: 2 Cores + 2 VPU/core + 1 MB L2
- Memory: MCDRAM: 16 GB on-package; High BW
- DDR4: 6 channels @ 2400 up to 384GB

KNL AVX-512



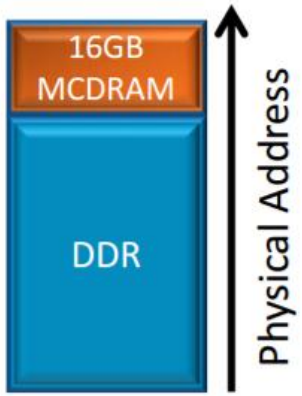
- 512-bit FP/Integer Vectors
- 32 registers, & 8 mask registers
- Gather/Scatter





http://geekswithblogs.net/akraus1/archive/2014/04/04/155858.aspx

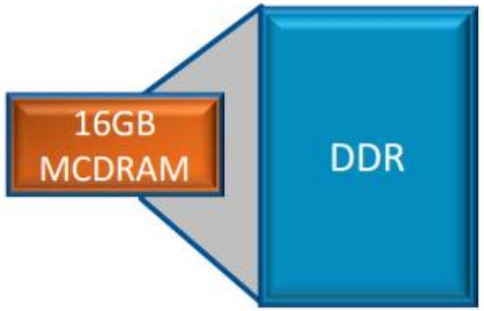
KNL MCDRAM



Flat mode

- MCDRAM as regular memory
- Managed by SW

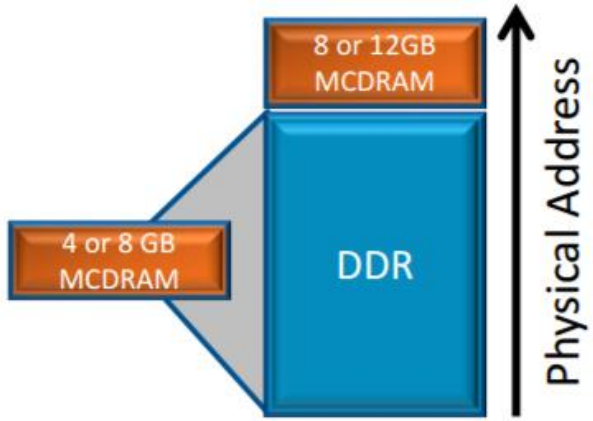
```
float *fv;
fv = (float *)hbw_malloc(sizeof(float) * 100);
```



Cache mode

- MCDRAM as cache
- No code change required*

```
float *fv; fv = (float *)malloc(sizeof(float)*100);
```



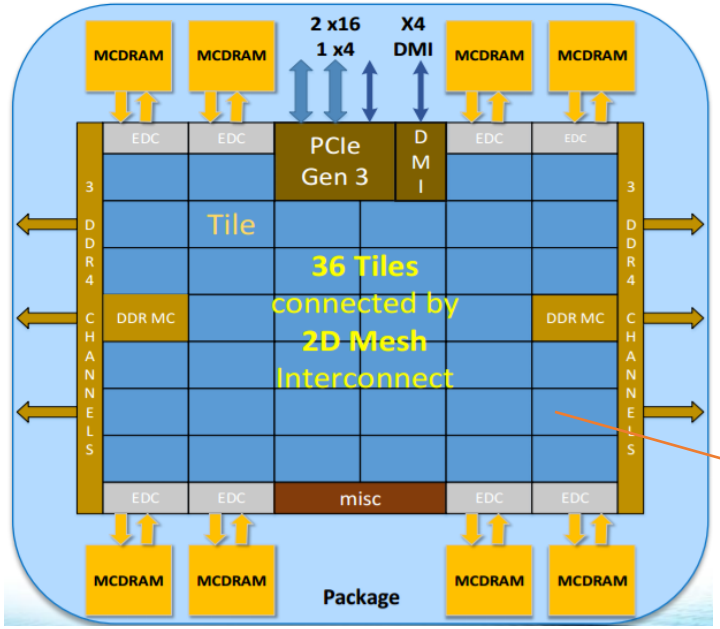
Hybrid mode

- MCDRAM part cache, part memory; 25% or 75% as cache

```
numactl -m 1 ./myProgram
```

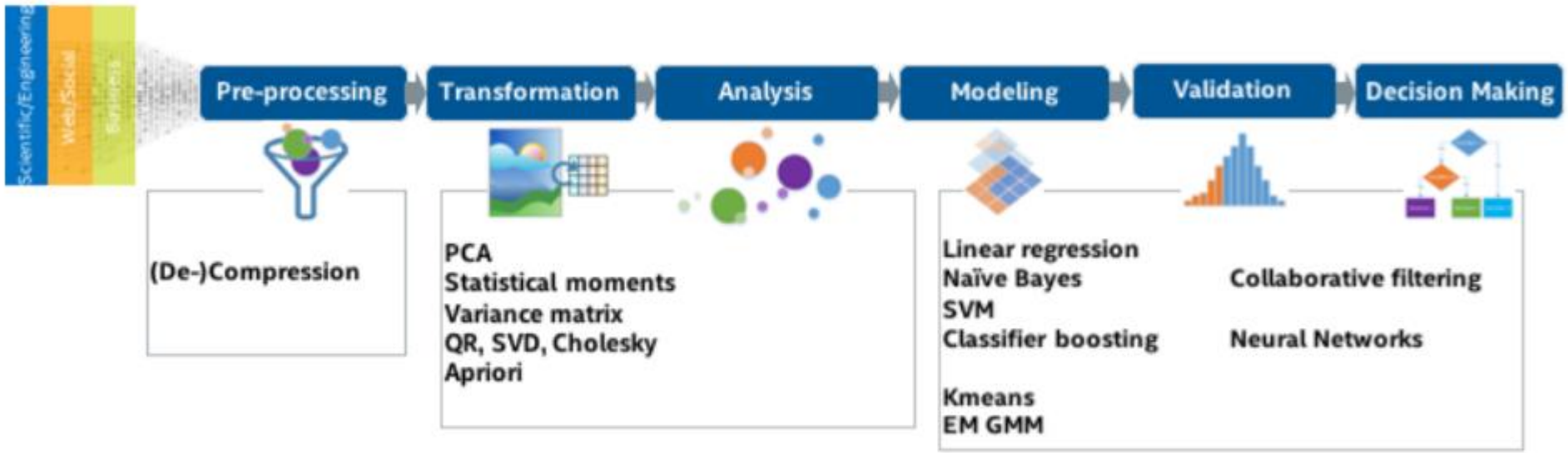
KNL Cluster Modes

- All-to-all: uniformly distributed address
- Quadrant: four vertical quadrants
- Sub-NUMA Clustering (SNC): each quadrant as separate NUMA domain



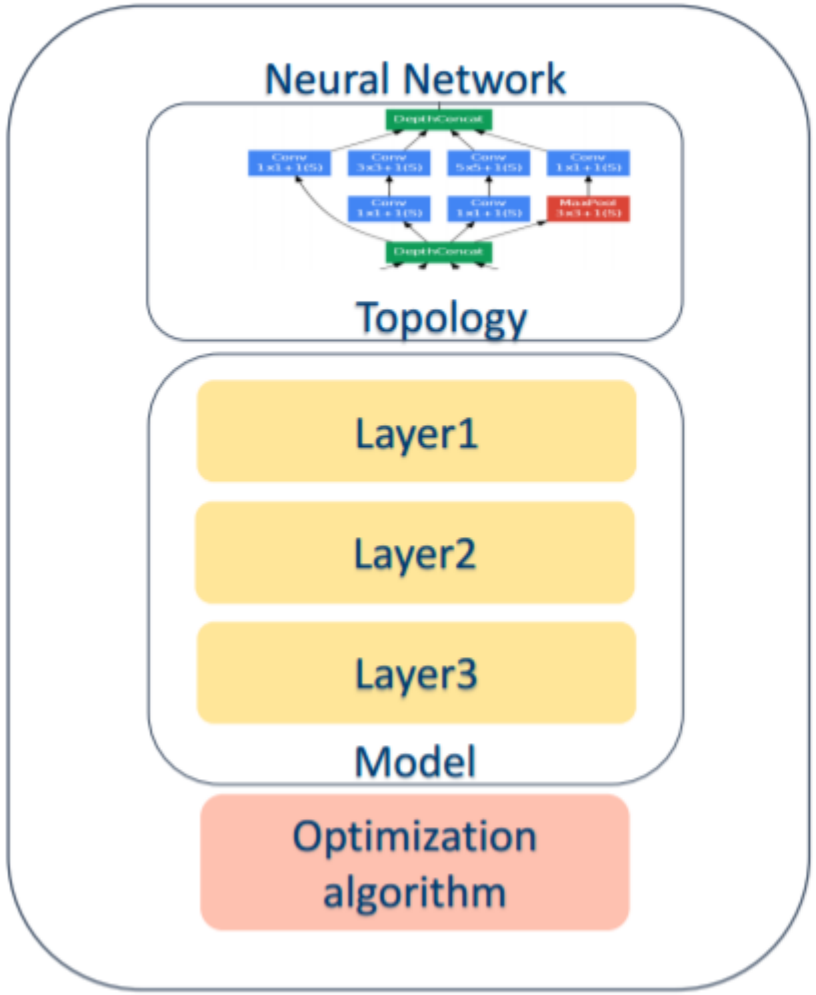
Tiles can be organized in 3 different cluster modes

Intel Data Analytics and Acceleration Library (DAAL)



- Optimized functions for deep learning and classical machine learning
- Language API for C++, Java and Python for Linux and Windows
- Support data ingress from Hadoop and Spark
- Free and open-source versions available

Intel DAAL



- **Layer**: NN building block
- **Model**: Set of layers
- **Optimization:** Objective function /solver
- **Topology:** NN description
- **NN:** Topology, model & optimization algorithm
- **Tensor:** Multidimensional data structure

Common layers	Activation	Normalization	Optimization / Solver
Convolutional	Logistic	Z-score	MSE
Pooling (max, average)	Hyperbolic tangent	Batch	Cross entropy
Fully connected	ReLU, pReLU, smooth ReLU	Local response	Mini batch SGD
	Softmax		Stochastic LBFGS
Dropout	Abs		

DAAL API Example

- Layer

```
SharedPtr<layers::fullyconnected::Batch<> > fcLayer1(new fullyconnected::Batch<>(20));
```

- Topology

```
SharedPtr<layers::fullyconnected::Batch<> > fcLayer1(new fullyconnected::Batch<>(20));
Collection<LayerDescriptor> configuration;
configuration.push_back(LayerDescriptor(0, fcLayer1, NextLayers(1)));
```

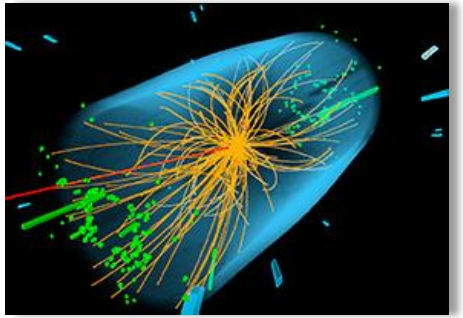
- Optimization Solver

```
services::SharedPtr<optimization_solver::mse::Batch<double> > mseObjectiveFunction(new
optimization_solver::mse::Batch<double>(nVectors));
optimization_solver::sgd::Batch<> sgdAlgorithm(mseObjectiveFunction);
```

- Model

```
trainingNet.compute();
… … ….
services::SharedPtr<training::Model> tModel = trainingNet.getResult()->get(model)
services::SharedPtr<prediction::Model> pModel = tModel->getPredictionModel();
```

Higgs Classification



Data

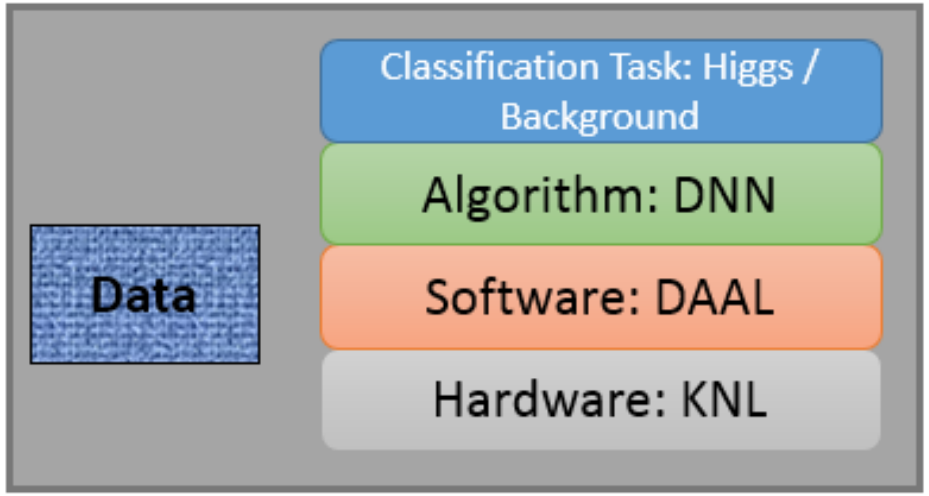
- 11 million events *(Monte Carlo simulations)*
  - 21 low-level features from particle detector
  - 7 high-level features (hand-crafted)
  - “1”: signal; “0”: background
  - A binary **classification problem**
- Training set: 10.5 million
- Validation set: 500 thousand

DNN

- Started with a topology with 3 layers (28-1024-2)
- Hyper-parameter: began with Random Search with minimal optimization effort

Model Development (DAAL)

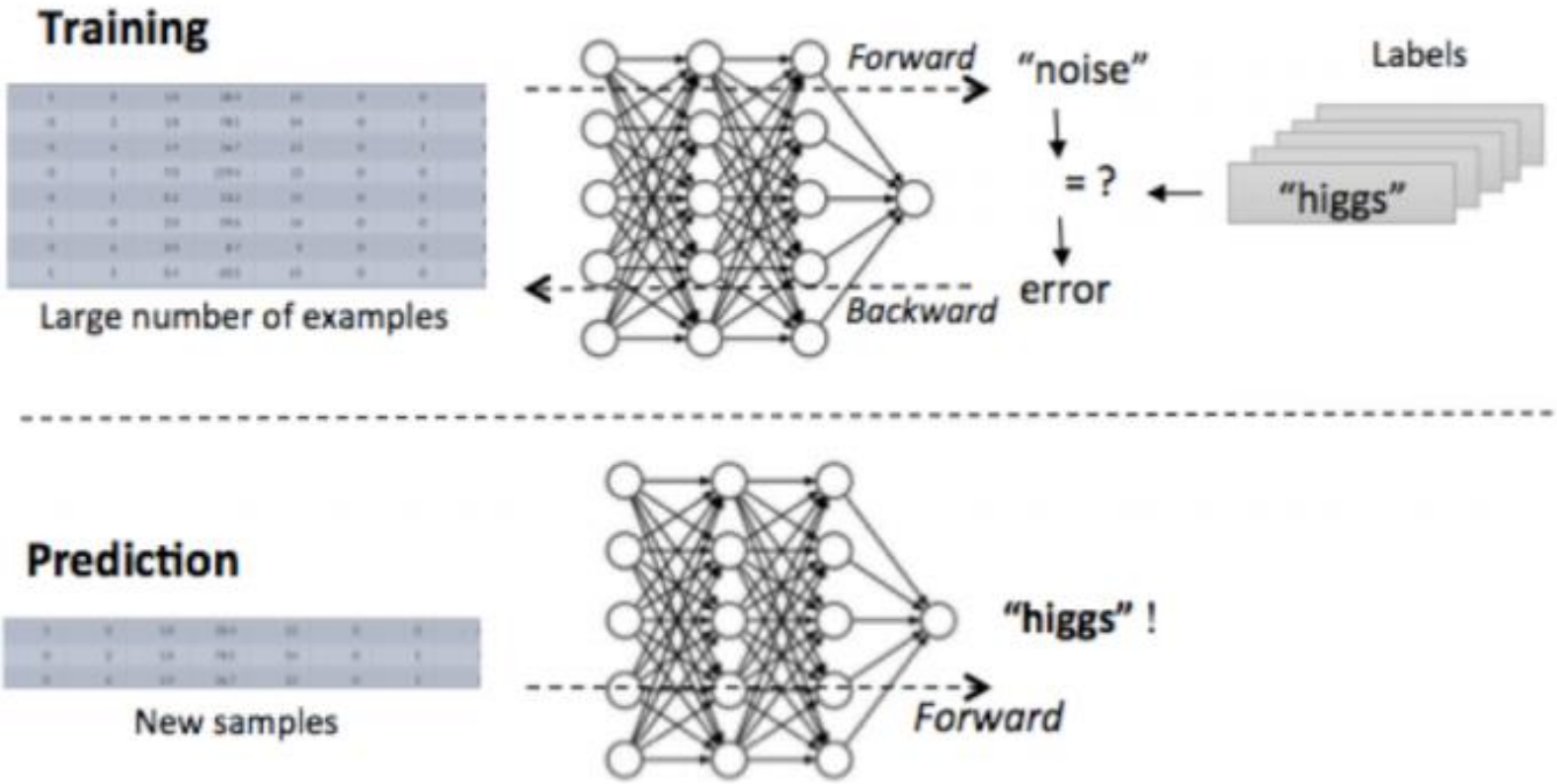
- Our “simulation” environment



KNL

- MCDRAM mode = Flat
- Cluster mode = Quadrant

DNN Topology



Preliminary Results

```
[davido@knl-data dkn]$ build/neural_net_dense_batch.exe ~/dataset/higgs/8g
Training started...
Prediction started...
Neural network classification results (first 20 observations):
Ground truth   Neural network predictions: each class probability
0              0.639  0.361
1              0.278  0.722
1              0.291  0.709
0              0.468  0.532
0              0.469  0.531
1              0.626  0.374
0              0.430  0.570
1              0.452  0.548
1              0.366  0.634
0              0.513  0.487
1              0.554  0.446
0              0.664  0.336
0              0.570  0.430
0              0.453  0.547
1              0.653  0.347
1              0.224  0.776
0              0.360  0.640
0              0.379  0.621
0              0.397  0.603
1              0.573  0.427
```

code: git clone https://github.com/davenso/DKN

Use and improve!

Discussions and Conclusions

- Performance can greatly enhance with:
  - Deeper network topology
  - Better hyper-parameters
- Deep neural networks are capable of learning underlying features, and should therefore generalize well, e.g. Higgs, Muon, etc.

Current Developments

- Exploration of **more complex models** and hyper-parameter optimization techniques (beyond Random Search)
- Integration of **“real” muon data** and performance benchmarking
- **Tuning of KNL hardware** to improve runtime performance of DNN training
- Implementation of distributed DNN algorithm, utilizing **multiple KNL nodes** for training
- Exploration of alternative algorithm (likely as a **‘hybrid model’**), e.g. Decision Forests

Thank You, UF Team

- Sergei Gleyzer
- Brendan Regnery
- Darin Acosta
- Ann Gordon-Ross
- Pranav Goswami
- Andrew Carnes
- Erik Deumens
- Jon Akers
- UF RC

Image credits

- http://www.nltk.org/book/ch06.html
- CERN
- http://www.cs.tau.ac.il/~wolf/OR/