Neural Network Exchange Format
Deploying Trained Networks to Inference Engines

Gergely Debreczeni, Chief Scientist
Outlook

• The NN deployment problem and its resolution
• About the NNEF working group, AIMOTIVE
• NNEF design philosophy
• NNEF components and usage
• Future directions and contribution
NNEF In a Nutshell - The problem

- There is a wide range of open-source deep learning frameworks available
  - Caffe, Torch, Theano, TensorFlow, Chainer, CNTK, MXNet, Caffe2, PyTorch
  - Each framework has its own model format to store trained networks

- Various chip vendors have released or are planning to release deep learning inference kits / engines
  - Nvidia, Intel, AMD, ARM, Apple, Qualcomm, ...

- Inference engines need to be compatible with many deep learning frameworks

- Network descriptions have no clear semantics (ambiguities)
NNEF In a Nutshell - The solution

- Create a unified network description format to facilitate deployment of networks from frameworks to inference engines
  - Describe network structure and data with clear semantics
- Provide tools to convert from frameworks to the exchange format
- Provide tools for inference engines to import the exchange format
  - No need to worry about where the network was trained
- Focus on Edge devices in production environments
  - Low power, low cost
  - Safety critical

Caffe
PyTorch
TensorFlow

NNEF

Vendor A
Vendor B
Vendor C
About the NNEF group

- Neural Network Exchange Format Working Group was founded in September 2016
  - Initiated by Almotive
  - After an exploratory phase earlier to investigate industry requirements
  - The standardization idea was also circulated among framework developers

- NNEF group is in collaboration with the OpenVX Working Group
  - OpenVX provides an execution model for running computational graphs on embedded HW for vision
  - Has a neural network extension, incorporates NNEF import

- Provisional specification was released in December 2017
- Version 1.0 was released in November 2018, revision 1.0.1 in April 2019
- Open for new companies to join!!
About Almotive

- Almotive is a software company delivering artificial intelligence based software stack for self-driving cars
  - Software components for recognition, localization, control
    - Relying primarily on camera inputs
  - Hardware IP for custom chip to accelerate neural networks in a low power budget with high efficiency

- Solutions heavily build on neural networks
  - We use various deep learning frameworks to train networks
  - We use GPUs and FPGAs for prototyping, custom chips for production
  - We experience the NN exchange problem in-house and in relation with partners
Deep Learning Frameworks - Similarities and Differences

• We work with and examined various frameworks
  - Torch, Caffe, TensorFlow, PyTorch (examined Theano, Chainer, Caffe2)

• They vary in the way they build networks, but the underlying operations are very similar
  - Most of the core ops are powered by the same implementation (cuDNN)
  - They build a computational graph that is similar on the lower level
    - The high level interface is different

• However, there are critical differences in the operations
  - Differences in parameterizations of computations (mathematical formulas)
  - Differences in output shape computations (asymmetric padding)
  - Differences in output value computations (border handling, image resizing)
NNEF Design Philosophy

- Convey all relevant information from DL frameworks to inference engines
- Platform independence
  - No hardware specification, no hardware specific data formats, etc.
- Flexible, extensible description (rapidly changing field)
  - By vendor specific operations
  - By future use cases and operations
- Easy to consume by compiler/optimization tools
- Implementable and optimizable on various hardware platforms
  - Hierarchical description, multiple levels of granularity
- Support for quantization techniques
NNEF Design Philosophy - Supported Network Architectures

• Support the following learning tasks
  - Image classification
  - Semantic segmentation
  - Object detection, instance segmentation
  - Video processing (action classification)

• Support at least the following network architectures
  - Fully connected networks (MLPs, auto-encoders)
  - Convolutional networks (feedforward, encoder-decoder)
  - Recurrent networks (LSTMs, GRUs)

• Support inference mainly, but training graphs are also possible
  - Needs extra operations for gradients/optimizer
NNEF Design Philosophy - Validation of Network Description

• Ensure that a network description can be easily validated
  - Syntactic/semantic validity of a document
  - Validity of the resulting graph
    - Implementation independent aspects
      - Graph connectivity, declaration of used identifiers
      - For example well defined tensor shapes and proper initialization

• Possibility to check that an inference engine can execute a network
  - Without loading the whole network
    - Structure is separated out from the data
  - Whether all operations/parameterizations are supported
What is included in the standard

- NNEF aims to abstract out the network description from frameworks
  - Only the network structure and data (no data feeding or training logic)
- A distilled set of frequently used operations
  - Well defined input-output mapping (output shape and value)
  - Well defined parameterization (math formulas)
- A simple syntax for describing networks on a medium level
  - Functional description, functional language-like
- Data format (binary) for storing network parameters (weights)
- Support for describing quantization info
NNEF Components - Structure description

• Devised a simple language to describe network structure
  - Describe a computational graph on tensor objects
  - Simple syntax, limited set of features
  - Strictly typed, single-static assignment, easier to analyze/validate

• Supports the hierarchical description of graph fragments
  - Similar to functions in scripting languages for graph building
  - Define larger fragments (compound ops) from smaller ones (primitives)
    - Instantly extensible with new compounds that can be built from primitives
    - Vendors don’t need to implement all primitives, can optimize compounds

• A predefined set of primitive and compound operations for building networks
  - Element-wise, activation, linear, pooling, normalization
NNEF Components - Data storage

• The structure description has data parameters (network weights)
• Parameter tensors are stored in a separate format
  - Simple data-format to store tensor data in floating point or quantized representation
• All the data and structure description is wrapped around with a container
  - Optionally compressed/encrypted tar file
  - Results in a single data-stream
NNEF Components - Quantization info

• Quantization is a crucial element of executing networks efficiently on embedded hardware

• Quantization information needs to be stored in the network description
  - In a platform independent manner
    - No reference to underlying data representations, like bit widths, arithmetic precision, etc.
    - Approach: ‘pseudo’ quantization, conceptually on real-valued data

• Quantization algorithms are various
  - Describe them as compounds built from primitives
    - Rounding and clamping operations

• Quantization algorithm for activations and for stored parameters
  - The data itself may be stored in the quantized format
  - Along with quantization algorithm
NNEF Generation and Consumption

- It is possible to write third party converters for DL frameworks
  - We have done that for Caffe, TensorFlow, PyTorch
  - Starting from proprietary formats of frameworks
  - Map operations to NNEF operations
  - Reverse conversion is also possible
  - Only for operations supported by both the framework and NNEF

- NNEF can be consumed by compiler stacks / proprietary inference engines
  - APIs may implement a subset of NNEF operations
  - NNEF operations may be lowered before consumption
    - The recipe for lowering can be defined in NNEF syntax
  - Compilation may happen online or offline
NNEF Tools

• Continuously developing a library of tools to support the usage of NNEF
  - github.com/KhronosGroup/NNEF-Tools

• File format parser (C++ and Python)
  - Easy to use, load/validate model in one line, return simple model structure

• Converter tools (Python)
  - Simple library to support writing of converters with a common logic
  - Available for TensorFlow, Caffe, ONNX, bidirectional conversion

• Model zoo: collection of models converted to NNEF for reference

• Future possibilities (contributions are welcome)
  - Quantization helpers
  - Graph optimization/visualization
  - Front ends for compiler stacks
Future Directions - Conformance

• NNEF is a file format which may be input to a compilation process
  - Compiler verification is well developed in practice
  - The file itself describes an executable graph

• Define NNEF conformance tests like compiler verification tests
  - Test cases separated by domain (image processing) and task (classification)
  - Test cases described by source files and input data
    - Source files are NNEF models
    - Input/expected output data represented as tensor binaries
  - Test cases cover a subset of operations
    - Subset of possible parameter combinations
  - Test cases for complete networks
    - Define task dependent metrics for evaluation, leave room for approximations
NNEF vs ONNX

• In our view, there are no clear conceptual advantages/disadvantages of one or the other

• Technical differences
  - ONNX uses binary format (protobuf)
  - NNEF uses text format for describing network structure for transparency

• Difference in development
  - ONNX is more rapidly changing, good for R&D
    - Developed by open source community
  - NNEF is a slower changing, more stable standard, better for industry
    - Developed by a consortium of industry players with a well established governance model
    - Member companies can have a vote on development decisions
NNEF Advisory Panel

- Anyone who wishes to review the NNEF specification draft can join an Advisory Panel
  - After signing and NDA with Khronos Group
- Provides early access to specification drafts
- Share feedback on mailing list
Thank you!

Contact Info

Gergely Debreczeni  
Chief Scientist  
AIMOTIVE  
Budapest, Hungary / Mountain View, CA  
gergely.debreczeni@aimotive.com  
www.aimotive.com  

Khronos Group  
www.khronos.org
Hands-on Demo

- Clone NNEF-Tools repository from GitHub
  - Install nnef python module
  - Few words about the organization of the repo (parser, io, converters)
- Show validator.py and sample.py
  - Turn on shape inference, compression
  - Change something in the examples to make it fail
- Show the model zoo
  - Download some ONNX or TF models and convert them to NNEF
  - Convert back to NNEF
- Show conversion from TF python code
  - Show the mapping of ops from TF to NNEF