Neural Network Exchange Format
Deploying Trained Networks to Inference Engines

Viktor Gyenes, specification editor
Outlook

- The NN deployment problem and the proposed solution
- About Khronos, NNEF group, AIMOTIVE
- NNEF design philosophy
- NNEF components
- NNEF usage, impact
- Release schedule and feedback
- Future directions
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 cuDNN/GIE, Apple DLKit, 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 with clear semantics
  - Describe network data
- Let frameworks convert their representation to the exchange format
- Let inference engines import only the exchange format
  - No need to worry about where the network was trained
About Khronos Group

- Khronos Group is an international organization, an open consortium of leading hardware and software companies creating Open Standards
  - Everyone is welcome to join, various membership levels
- Has authored various Royalty Free specifications
  - Parallel computing, graphics and vision, sensor processing
    - OpenGL, OpenCL, OpenVX, Vulkan, ...
  - Independent but cooperating Working Groups for each standard
- Focus is on hardware acceleration
  - Embedded devices, edge devices, mobile
  - CPUs, GPUs, FPGAs, SoCs, ...
- Conformance tests and adopters program for specification integrity
  - Cross vendor compatibility
About the NNEF group

• Neural Network Exchange Format Working Group was founded in September 2016
  - Initiated by Almotive
  - After an exploration that began in early 2016 to investigate industry requirements
  - The standardization idea was also circulated among DL framework developers

• NNEF group is in collaboration with the OpenVX group
  - OpenVX is an image processing API with a neural network extension
  - Provides an execution model for running neural networks on embedded devices
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 chips to run neural networks in a low power budget with high efficiency

- Heavily builds on neural network solutions
  - 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 examined and worked with various frameworks
  - Torch, Caffe, TensorFlow (examined Theano, Chainer, Caffe2, PyTorch)
- 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, often use scripting languages
- 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 engines/libraries/drivers written in low level languages
  - Scripting languages are often not available in embedded environments
• Implementable and optimizable on various hardware platforms
  - Hierarchical description, multiple levels of granularity
• Support for quantization techniques
NNEF Design Philosophy - Supported Network Architectures

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

• Support the following learning tasks
  - Image classification
  - Semantic segmentation
  - Object detection, instance segmentation
  - Language processing (syntactic analysis, sentiment analysis)
  - Audio processing
  - Video processing (action classification)
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
    - 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
  - 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-low level
  - Very high level scripting is not priority, too many options, hard to standardize

- Data format for storing network parameters (weights)

- Support for describing quantized networks
NNEF Components -
Structure description

• Devised a simple language to describe network structure
  - Python-like syntax, limited set of features
  - But strictly typed, easier to validate

• Supports the hierarchical description of graph fragments
  - Similar to Python functions 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)
  - Typically named in a hierarchical fashion according to network structure
    - For example variable scopes

• Parameter tensors are stored in a separate format
  - Simple data-format to store tensor data in floating point or quantized representation
  - Organized into hierarchy according to scopes

• All the data and structure description is wrapped around with a container
  - Results in a single data-stream
  - May provide compression or encryption
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 for Deep Learning Frameworks

- It is possible to write third party converters/exporters
  - We have done that for Caffe and TensorFlow
  - Need to map structure description and parameter data to NNEF

- It would be good to have ‘built-in’ support for NNEF export in deep learning frameworks
  - Needs to be maintained when frameworks evolve (may happen frequently) or when the standard is updated (happens rarely)

- Importing NNEF to DL frameworks would be possible when all features are supported by the framework
  - Reverse conversion tools
NNEF for Inference Engines

• APIs may choose to implement a subset of ops
  - Or even a subset of parametrizations for a given op

• APIs may choose which ops to treat as atomic and which as compound
  - Provide optimized compounds

• APIs may choose to compile NNEF offline into a vendor specific description
  - Or consume NNEF directly

• NNEF does not define conformance tests
  - Only the ideal case of infinite arithmetic
  - When is an implementation accurate enough?
    - Can’t be defined without reference to data representation (platform dep.)
    - Want to leave room for fast approximations at the cost of accuracy
    - OpenVX will define execution model and conformance tests using NNEF
Usage Scenarios, Impact

- The most important target use case is deployment of trained networks to inference engines

- Further use cases may include
  - A common format for network conversion tools
    - High level graph transformations and optimizations
    - Quantization
  - It would also facilitate transferring networks among DL frameworks...

- Further impact may include
  - Research results exported into NNEF would become immediately executable on various hardware (portability) and possible to integrate into various applications
  - May drive DL frameworks to be even more compatible with each other
Planned Release Schedule

- Start with a Provisional Release
  - Opportunity to take community feedback into account before finalizing the release
  - Scheduled to be released before the end of 2017
- The final release would probably arrive after a period of public feedback
  - Around mid 2018
- Subsequent releases would contain improvements as necessary
  - Syntactic features
  - New operations
  - Support for training
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 or teleconferences organized on-demand
Future directions - Training

• At its core, the training process is just another recurrent computation
  - Many complicated aspects: support at least the computational aspect
  - Ignore data-feeding, parameter tuning, validation, etc. aspects

• The primitive ops are very similar
  - Backward ops are often used in feed-forward networks as well (deconv)
  - Solvers and regularizers are possible to describe from primitives
  - Initializer ops can be introduced

• Generate training graph with automatic differentiation from inference graph
  - Either in the DL frameworks before export (e.g. Caffe2)
  - Or in NNEF by third party graph conversion tools
Thank you!

Contact Info

Viktor Gyenes, PhD
Lead AI Research Engineer
AIMOTIVE
Budapest, Hungary / Mountain View, CA
viktor.gyenes@aimotive.com
www.aimotive.com

Khronos Group
www.khronos.org