**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11/N17740**

**April 2018, San Diego, CA, USA**

**Title: Use cases and requirements for compressed representation of neural networks**

**Source Requirements Subgroup**

**Status: Approved**

# Introduction

## Motivation

Artificial neural networks (NNs) have been adopted for a broad range of tasks in multimedia analysis and processing, media coding, data analytics and many other fields. While the underlying technology has been known for decades, the recent success is based on two main factors: (1) the ability to process much larger and complex neural networks (deep neural networks, DNNs) than in the past, and (2) the availability and capacity of large-scale training data sets. These two aspects not only make trained networks powerful, but also mean that they contain a large number of parameters (weights), resulting in quite large sizes of the trained NNs (e.g., several hundred MBs).

In many cases, these NNs are used to replace certain components in a processing workflow, that have previously relied on handcrafted approaches, e.g. for feature extraction or filtering, and NNs have been shown to outperform these handcrafted approaches. The correct execution of the workflow thus requires the deployment of a particular trained network instance, which is then evaluated as part of the workflow (a step called inference). Inference may be performed on a large number of devices (e.g., in consumer applications), thus requiring to transmit the trained NN over a network connection, and possibly these devices may have limitations in terms of processing power and memory (e.g., on mobile devices or smart cameras).

The NNs used in an application can be improved incrementally (e.g., training on more data, including feedback from validation of results), so that updates of already deployed networks may be necessary. In addition, the NNs for many applications (e.g., classification) start from an NN that has been pretrained on a general dataset, and then adapted and retrained for the specific problem. Thus, different applications may use NNs that share large parts among them.

In existing work on neural network compression, it has been shown that compression factors in the range of 10-100x are feasible, with no or only small impact on the performance of the NN in a particular use case. As the description of the network topology is rather small compared to the parameters/weights, compression technology will in particular address compression of weights, e.g., by reducing their number, quantising them, representing them more compactly etc.

Any use case, in which a trained neural network (and its updates) needs to deployed to a number of devices, which potentially run on different platforms or in applications of different manufacturers, could benefit from a standard for the compressed representation of NNs. Compression will enable an application to have smaller representations of NNs sent across network connections, and potentially also NNs having smaller memory footprint during inference. While exchange formats for NNs exist (e.g., ONNX, NNEF), they do not yet address compression and incremental updates. A standard for the compressed representation of NNs will ensure interoperability with inference environments on different platforms.

## Scope

MPEG NNR aims to define a compressed, interpretable and interoperable representation for trained neural networks. MPEG NNR shall be able to

* represent different artificial neural network types (e.g., feedforward networks such as CNN and autoencoder, recurrent networks such as LSTM, etc.)
* enable efficient incremental updates of compressed representations of NNs
* enable scalability, i.e. NNs of different performance can be obtained from subsets of the complete compressed representation
* inference with compressed network
* enable use under resource limitations (computation, memory, power, bandwidth)

The scope of existing exchange formats (NNEF, ONNX) is the interface between the framework used for training and the acceleration library/optimisation engine for a specific platform.

If we consider a use case of deploying a trained network to a large range of target devices (e.g., mobile phones, signal processors in a vehicle), the process is likely to have these steps (see Figure 7):

* Training of the network with deep learning framework *L*,resulting in trained neural network *T*
* Export to an exchange format (neural network *T’*)
* Optimisation for 1,..,*n* target platforms, using acceleration/optimisation libraries *A*1, …, *An*, resulting in optimised neural network *O*1, …, *On*
* Distribute the networks *Ok* to the terminal devices, where they are executed with specific (software or hardware) inference engines *Ik*
* Alternatively, inference may be performed in an inference engine I\* directly from the exported network, without using an accelerator library



Figure 1: Deployment/inference process.

We can make the following observations about the two interfaces:

* Compression based on a standard format would target *T’*, which is not where the volume of data is distributed
* Compression would be most beneficial for *Ok*, where networks are distributed to a large number of devices running the inference engine.
* *Ok* may already have a pruned network structure and quantised weights, so that compression is less (in the worst case not) effective
* *Ok* may not be easy to represent in a standard representation, and the format coming out of a specific acceleration library *Ak* may not be documented at all

Some of the MPEG NNR use cases would benefit from compression on the first interface, while for other only a compressed representation for the second interface would be beneficial.

A processing chain for neural network compression is shown in Figure 2. Some of the steps change (simplify) the model structure, while others affect only the weights. Several of these operations are also performed by the acceleration/optimisation libraries for specific target platforms.

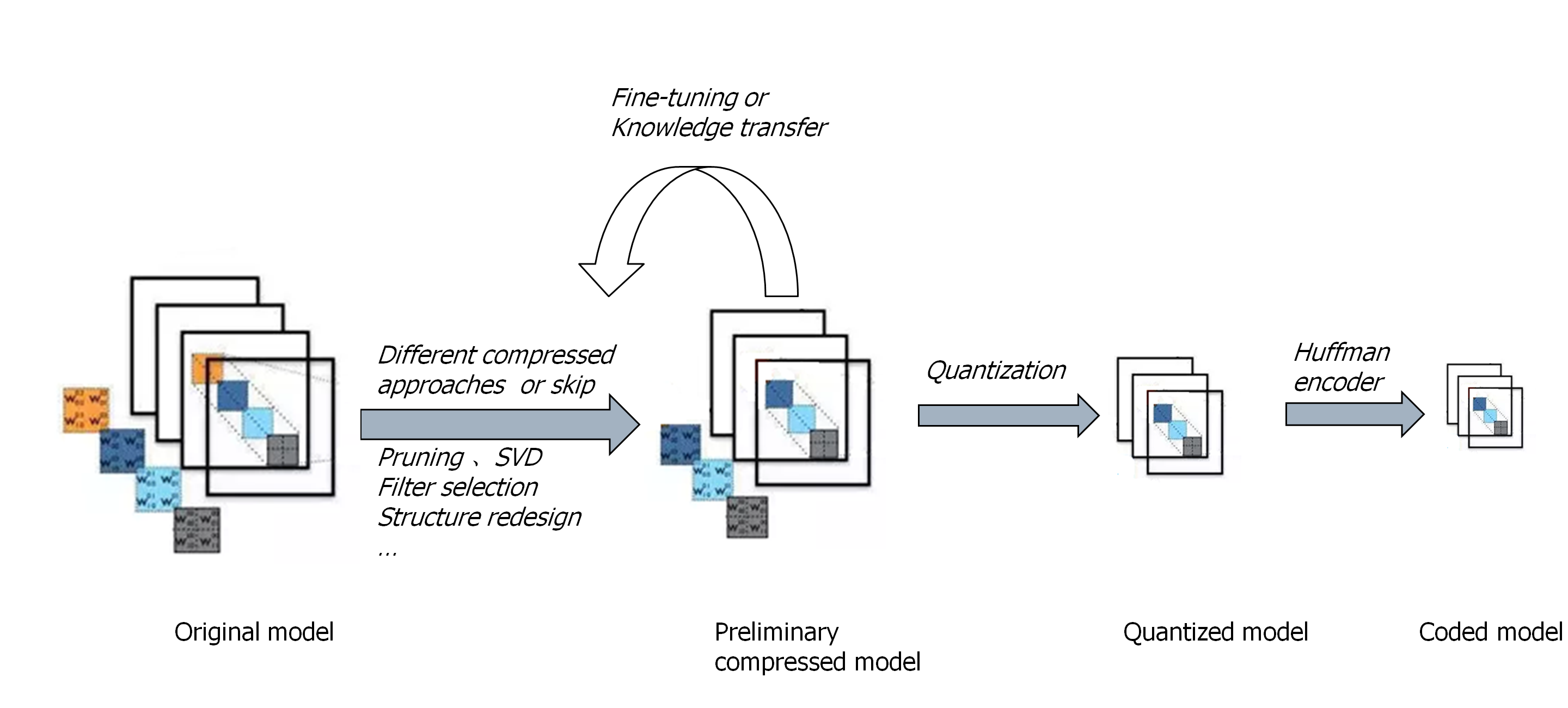


Figure 2: Framework for neural network compression.

The remainder of this document groups the proposed use cases and provides an overview of the requirements from these use cases.

Somme use cases have been assigned to one of the groups for structuring this document, but may include features that belong to multiple groups, e.g., include training and distribution, or include training and processing.

# Use cases on NN distribution/deployment

|  |
| --- |
| UC1 Installing NN-based applications |
| In an extension of the current app model, users install Neural Network-based applications on their devices. The NN-based core of the application can be compressed |
| **Overlap with other use cases** |
| General description of use cases for distribution/deployment, other UCs in this category can be seen as specialisations |
| **Required  features** |
| 1. Lossy compression 2. Scalability (if end user stops downloading compressed NN, a level of performance is still guaranteed) |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The device converts the compressed representation and obtains a NN capable of processing data.  We need to explore whether the compressed representation can process data at what cost |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)? What is the estimated size of these neural networks?** |
| At steady state this scenario would envisage tens of millions downloads every day globally, possibly growing to hundreds of millions.  The size of neural networks will be application dependent |

|  |
| --- |
| UC2 Camera app with object recognition |
| Recent smart phones include cameras, that adjust their automatic mode based on scene/object recognition results. For example, the Huawei Mate 10’s camera app classifies 13 scene/object types[[1]](#footnote-1).  In order to add/improve classification, the trained NN needs to be updated in the app. There are probably also cases where users are interested in adding types of objects/scenes, training some on their own and transferring models from one phone to another. |
| **Overlap with other use cases** |
| Specific example of UC1 and UC5, due to limitations of mobile device UC6 is relevant  Overlap with UC10 if retraining with local images is considered |
| **Required features** |
| * Significant size reduction of the trained network to speed up download of updated NNs. * If lossy compression is used, the performance of using a compressed representation of the network must be similar to that of using an uncompressed network. * (optional) The use of the network should be possible without inflating it to its original size in memory. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| Currently, the app and the trained classes come from the same provider. Huawei has stated that there are plans to add more object classes and use user data to train classes. Third parties could provide trained NNs, and exchange of trained NNs would be necessary to avoid lock-in of users who invested in training their own models. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Updates are expected to be infrequent (maybe a few times per year), but potentially to a large number of devices. * For ImageNet (1,000 classes), trained NN sizes are in the rage of 50-500 MB. * For mobile devices, the download bandwidth may vary considerably due to the network type. * Latency is an issue when a user downloads a new model to use it right away. |

|  |
| --- |
| UC3 Translation app |
| Some translation apps (such as e.g. Microsoft Translator[[2]](#footnote-2)) make use of NNs for language recognition and/or synthesis. Those apps can be extended with a set of languages, that can be downloaded within the app. The data for a language contains the trained networks for a language, but also other data (e.g. dictionaries). |
| **Overlap with other use cases** |
| Specific example of UC1, due to limitations of mobile device UC6 is relevant |
| **Required features** |
| * Significant size reduction of the trained network to speed up download of additional languages. * If lossy compression is used, the performance of using a compressed representation of the network must be similar to that of using an uncompressed network. * (optional) The use of the network should be possible without inflating it to its original size in memory. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| Currently, the app and the trained language models come from the same provider, and are designed to run on a specific set of mobile platforms.  A standard representation would allow third parties to offer additional, or specifically trained models (dialects, terminology) for the translation app. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Updates are expected to be infrequent (maybe a few times per year), but potentially to a large number of devices. * The size of the data per language (not only the NN) is currently around 200MB. * For mobile devices, the download bandwidth may vary considerably due to the network type. * Latency is an issue when a user downloads a language to use it right away. |

|  |
| --- |
| UC4 Large-scale public surveillance |
| |  | | --- | | In recent years, surveillance cameras in public area are increasing and it is becoming a challenging problem to realize automatic surveillance systems for public areas such as stations, large-scale parks, exhibition halls, stadiums and shopping malls. Video analysis using deep neural network is becoming one of essential core functions for automatic surveillance systems.  Fig 1 shows one of potential automatic surveillance systems. Area 1 to area M have local automatic surveillance system composed of surveillance cameras which have a function to extract metadata like the output of object detection, object tracking, action recognition, etc. by deep neural network and a local server as “surveillance AI” to recognize the situation of the corresponding area by the metadata from the cameras, respectively. A cloud server periodically re-trains the neural network for the metadata extraction on the surveillance cameras and provide it to the cameras. Since the automatic surveillance system can be connected to a huge number of cameras[[3]](#footnote-3), it will be desirable to compress the neural network when it is transferred to the cameras. Note that it may be necessary to transmit different networks to different areas, e.g., depending on the relevant tasks for each area.  Moreover, since it is expected that it will be realized to collect/generate the training dataset automatically for re-training of the neural network in the future, the update cycle of the neural network will become shorter. In consideration of such situation, the compression of the neural network will be more important to reduce the traffic over the network.    Figure 3: An example of future public surveillance system. | |
| **Overlap with other use cases** |
| Specific case of UC1 |
| **Required features** |
| * Low computational complexity for decoding the compressed neural network model * Low memory consumption for decoding the compressed neural network model * “High efficient” (means small memory sized) representation for decoded neural network model * High error robustness for compressed neural network model * Ability to detect the manipulation of the compressed neural network model |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| * Interfaces benefitting from the standard: Surveillance cameras * Who: Providers of the automatic surveillance systems |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Need to update the neural network once a month or less in near future and once a week or more in further future. * Over hundreds million surveillance cameras will be connected to the surveillance systems. * Latency of download the neural network is less important compared with that of execution of actual process by the neural network |

|  |
| --- |
| UC5 Visual pattern recognition (VPR) |
| Visual pattern recognition (VPR) techniques are an important component of intelligent system and are used for many application domains. Visual pattern recognition is one of the most effective applications to which deep neural network can be applied, and excellent recognition results have been reported in many recent studies. In particular, visual analysis and pattern recognition applications in mobile and IoT devices are becoming more popular. In order to apply DNN in a mobile and IoT environment in which computing capability is limited, a system configuration cooperating with a PC or server is considered as a practical way. In other words, a networks is trained in a PC or a server, then pattern recognition is performed using the trained network in a mobile/IoT device.  Fig.1 shows an example such case of VPR systems using DNN in which object detection is performed. A set of neural networks with different algorithms are trained and provided by PC/Server. A user can select an appropriate network for the given application of pattern recognition, then the network model and the trained results of the selected network are delivered to a mobile/IoT device in which pattern recognition is performed.  In addition, a user can also optionally report the evaluation of the pattern recognition result, which is feedback to the server in which network will be retrained/updated.    Fig. 1 An example of pattern recognition using DNN in an mobile/IoT device |
| **Overlap with other use cases** |
| Specific case of UC1, similar to UC2, due to the target device UC6 applies |
| **Required features** |
| * High efficient representation of the neural network representation and trained weights * (optional) The use of the network should be possible without inflating it to its original size in memory * (optional) Scalability of representation in terms of leaving choice of some parameters open |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| * Interface between a mobile/IoT device and a server can be supported in an interoperable way to deliver trained network for pattern recognition * Implementation of pattern recognition in a mobile/IoT environment with limited computational power. * a DNN based pattern recognition device can be provided by different vendors * Who: service providers (DNN provider), vendors |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Updates are expected to be infrequent (incremental training, different pattern recognition methods), but potentially to a large number of devices. * The size of the training set and the method used to build it are often dependent on the selected pattern classification technique. * Latency is in most cases not critical, except for cases where a user downloads the mobile app to use it right away. |

|  |
| --- |
| UC6 NN representation for devices with limited memory and bandwidth |
| This use case covers a scenario where the device that will run a neural network has very limited memory, computational capability and bandwidth. An example of such an application is object recognition in memory, computation and bandwidth limited IoT devices which are capable of media processing.  It is challenging for such devices to load large neural networks and do inference. It is also not always possible to compress a neural network so that it fits into the memory of the IoT device. Moreover, such a demanding compression usually has an impact on the inference performance (i.e. significant loss of accuracy). Hence, it is required that a mechanism is defined to handle loading and inference of such neural networks that run on devices with very limited memory, computation capability and connectivity bandwidth.  A group of such devices, connected together, form an processing chain of neural network layers, in which each devices may process one or more layers of computation.  This group of devices can be used at both the trainign stage and the inference stage.  . High-speed for fast processing, and hardwired connection for security motivations (optional)  Another need is to make up for layer-wise NN computation, in case some of NN layer communications are interrupted due to various reasons e.g. communication errors, unaccessible devices etc. |
| **Overlap with other use cases** |
| Specific case of UC1 (restricting the type of target platform), related to UC2, UC3 and UC5 |
| **Required features** |
| * A mechanism for the communication of NN models to memory and/or bandwidth and/or computation limited devices. * A communication protocol to facilitate this kind of mechanism. * A mechanism for guiding the execution of NN models. * In the case of distributed layers, it is needed to signal at initialization phase for group of NN devices to perform inference and training:   + Mechanism to declare device capabilities (e.g. available memory, battery status, computational capabilities, expected availability of resources)   + Mechanism to sends, to each device, parts of distributed NN specific for the device in a multicast or unicast fashion.   + Mechanism to exchange and signal parts of distributed NN architecture and weights. * At execution phase (inference or training stages), each device sends its outputs to following devices according to the information it received at initialization phase. * Singaling for the backup processing in case of interrupted layer-wise or filter-wise NN computation. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The use case is aimed for low-memory, low-bandwidth devices.  Required components are involved in mobile devices and memory and bandwidth limited IOT devices. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * The rate of neural networks transmission depends on the application use cases. * The amount of information to be communicated is determined by the bandwidth of the device that the neural network will be uploaded on. * Estimated size of the neural network can vary from hundreds of KBs to hundreds of MBs. * The expected bandwidth of the distribution channel from the minimal of 100Kbs. * The maximum latency is application dependent, in the range of mili-seconds to seconds. |

|  |
| --- |
| UC9 Efficient re-use of neural networks among different media applications |
| Many applications utilize neural networks derived from the same neural network. This is the case for example for mobile phone apps. The common procedure consists of taking a neural network pre-trained on a large dataset (such as ImageNet), which is able of extracting high-quality generic visual features, and deriving from it a new model for a specific down-stream task. The derivation may consist of one or more of the following options:   * fine-tuning all layers; * freezing some layers/weights and fine-tuning only the remaining layers/weights; * freezing or fine-tuning some layer, and adding new layers and branches which are trained.   This means that several apps would need to store on the device the same base neural network multiple times. This is a waste of both storage and inference speed. In fact, inference speed may become very low if multiple models need to be run on the same device at the same time, e.g., when taking a picture or video the camera may run a camera parameter tuning neural net, a person detection neural net, a style-transfer neural net, etc.  There is a need to efficiently share & reuse a network model among multiple applications and tasks.  Another need related to the use case is the update (versioning) of the neural network that an app use. The versioning can either be needed to upgrade the network to a better performing one (tested in the server side).  Still another need is the layer-wise or filter-wise updating of the neural network that an app use in each stream, a network layer (or convolutional filter) is sent to the device to perform designated computation on devices.  Another need is to make up for layer-wise NN computation, in case some of NN layer communications are interrupted due to various reasons e.g. communication errors, unaccessible devices etc. |
| **Overlap with other use cases** |
| Extension of UC1 (partial updates of models), also applicable to other UCs in this group |
| **Required features** |
| * Signaling for negotiating the re-use of NN models among applications and between application providers and users. * Signaling for designated network layers or filters to be sent and performed on devices. * Signaling for the initialization and closing of such kinds of layer-wise / filter-wise NN computations. * Singaling for the backup processing in case of interrupted layer-wise or filter-wise NN computation. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The interfaces which will benefit will be the media consumption devices, which would require less bandwidth because less layers will be transmitted. Also, the devices will save storage as less neural networks will be stored on device. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| Frequency  - Every time a new model for a certain application or a new version of that model needs to be transmitted. For mobile phones, the frequency may be from every day to every month for each user.  Size  - From few KB to several hundreds MB  Bandwidth  - The bandwidth may depend on the specific distribution channel. One example is 2G, 3G and 4G networks.  Maximum latency  - Can be from few milliseconds to few seconds. So, it is application-dependent. |

|  |
| --- |
| UC14 Electronic Health Record and Genomic Data |
| The development of deep learning network (DNN) technology and the accumulation of big data have led to active research to utilize deep learning about medical information. The technology of diagnosing diseases by applying deep learning to medical images has been developed much, but recently, there is a growing interest in exploiting health record and genomic data for better healthcare by use of DNN technologies on individual health record and genomic data to better predict the correlation between disease and individual trait.  The efforts of life sciences researchers have accumulated various levels of information, from genomic data to medical information data, and open Databases are being provided. As an example, it is possible to use this data to develop algorithms for predicting disease risk by collecting and processing medical information big data centered on genome information of open databases such as GTEx, NCI-60, ENCODE, ICGC, 1000 Genome, NIH Epigenomics Project and GIANT [1].    Figure 1 Open databases about medical information [1]  Generally, genetic information and medical information data cannot be easily shared because of privacy violation. Therefore, it is necessary to define an intermediate neural network data exchange mechanism that can guarantee the privacy protection. |
| **Overlap with other use cases** |
| * Specific case of UC1. |
| **Required features** |
| * Use of weight calculation of the trained network to use in medical information. * Trained NN size should be low memory consumption. * Error resilience to provide high precision health recommendation. * Intermediate neural network data exchange mechanism that can guarantee the privacy protection. * Compression of network parameters (weights) is necessary. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| * It can be used regardless of the platform and software. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Once the initial NN deployment is completed, it will not happen frequently, but it will be updated again if there is a new set of training data. * The size of uncompressed NN model is about nearly GB up to TB depending on individual sample size and deep learning model. * The download bandwidth may vary considerably due to the network type. * Latency of download the neural network is less important compared with that of execution of actual process by the neural network. |

[1] Leung, Michael KK, et al. "Machine learning in genomic medicine: a review of computational problems and data sets." Proceedings of the IEEE 104.1 (2016)

|  |
| --- |
| UC15 Dynamic adaptive media streaming |
| Universal Media Access (UMA), as proposed in the late 1990s and early 2000s, is now reality. It is very easy to generate, distribute, share, and consume any media content, anywhere, anytime, on any device. These kinds of real-time entertainment services -- specifically, streaming audio and video -- are typically deployed over the open, unmanaged Internet and account for the majority of the Internet traffic. A major technical breakthrough and enabler was certainly the HTTP Adaptive Streaming (HAS) technique resulting in the standardization of MPEG Dynamic Adaptive Streaming over HTTP (DASH). According to Cisco’s Visual Networking Index (VNI), the global IP video traffic will be 82% of all IP traffic by 2021 (up from 73% in 2016). Interestingly, live Internet video will account for 13% of Internet video traffic by 2021 (it is expected that it will grow 15-fold from 2016 to 2021). Nielsen's Law of Internet bandwidth states that the users' bandwidth grows by 50% per year (i.e., 10% less than Moore's Law for computer speed), which roughly fits data from 1983 to 2018. Thus, the users' bandwidth will reach approximately 1 Gbps by 2021. Therefore, as media rates and network bandwidth continuously increase there’s a continuous need to provide support for dynamic adaptive media streaming in order to address the heterogeneity of today’s and future devices and networking infrastructure.    Figure 1. HAS in a nutshell.  HTTP Adaptive Streaming (HAS) including but not limited to Dynamic Adaptive Streaming over HTTP (DASH) allows for seamless delivery of media within heterogeneous environments over the top of existing infrastructure to enable high Quality of Experience (QoE). The QoE is mainly determined by factors like media bitrate/throughput, (startup) delay, stalls, switches, etc. and AI-based methods can help to improve the QoE as pointed out recently. As a consequence, we see a need to support dynamic adaptive media streaming within this new activity on coded representation of neural networks. In particular, the coded representation of neural networks needs to be made available to such clients (i.e., initially prior to streaming and updates during the streaming session) to enable better adaptation decisions and, thus, increase QoE. |
| **Overlap with other use cases** |
| Specialization of UC1 targeting dynamic adaptive media streaming. Also related to UC12 (coding) and UC 13 (processing) |
| **Required  features** |
| * Compression and partial delivery (aka streaming) incl. updates to the clients; possibly negotiation between server-client * Latency (initial and during the session) * Scalability wrt today’s and future infrastructures (CDN, ICN, SDN, 5G, etc.) |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| **Which is/are the interface(s) that benefit from a standard neural network representation?**   * Client (e.g., web/HTML5 browser, mobile/stationary devices running specific software) receiving the information from a “server” which might be located somewhere within the delivery environment (i.e., might also come from the “network”).   **Who is in control of the components (e.g., mobile devices, embedded processors) involved?**   * Content/service and/or network/infrastructure provider; basically, everyone within the delivery environment (see also DASH SAND model) |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**   * Initially during session setup and regular updates during the session, specifically for live (24/7) channels.   **What is the estimated size of these neural networks?**   * This depends on the actual application requirements but could be KBs to MBs.   **What is the expected bandwidth of the distribution channel?**   * The full spectrum ranging from Kbps to Gbps (and beyond).   **What is the maximum latency that is acceptable?**   * This depends on the actual application requirements but could be milliseconds to seconds. |

1. Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. 2016. Online learning adaptation strategy for DASH clients. In Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16). ACM, New York, NY, USA, Article 8, 12 pages. DOI: https://doi.org/10.1145/2910017.2910603
2. Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). ACM, New York, NY, USA, 197-210. DOI: https://doi.org/10.1145/3098822.3098843
3. Chaitanya Ekanadham, Using Machine Learning to Improve Streaming Quality at Netflix, Online, March 2018, https://medium.com/netflix-techblog/using-machine-learning-to-improve-streaming-quality-at-netflix-9651263ef09f
4. M. Gadaleta, F. Chiariotti, M. Rossi and A. Zanella, "D-DASH: A Deep Q-Learning Framework for DASH Video Streaming," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703-718, Dec. 2017.  
   doi: 10.1109/TCCN.2017.2755007

# Use cases on NN (re)training

|  |
| --- |
| UC7 Deep NN Factory |
| The scientific and technical advancements witnessed in the last 5 years in the field of machine learning based on deep neural networks is making a whole new business area grow at an incredible pace. Main cloud providers are offering increasingly engineered services that can be used by their customers to train specialized networks on demand. If on the one hand training such complex networks in a reasonable amount of time requires the availability of a non-trivial amount of dedicated hardware, which can be affordable only by said providers, on the other hand the execution (inference) phase can be accommodated on much simpler infrastructure, in most cases affordable by the customers themselves. Thus, the opportunity to “download” pre-trained networks and use them on site starts to be a meaningful use case in this domain. The necessity to retrain/refine networks as long as the statistics of the data evolve or new classes are needed is an additional element making this scenario even more realistic. A “Deep NN Factory” is therefore a system implementing such training/refinement service and producing pre-trained (or refined) deep networks on demand of its customers. Delivered NN by the Factory can be compressed using a lossless or lossy technique, depending on the requested delivery latency. This use case is limited to those factories operating in the multimedia domain, and in particular specialized in multimedia classification tasks. |
| **Overlap with other use cases** |
| Relation to UC10 (specific case of distributed training) |
| **Required  features** |
| * The Deep NN Factory shall accept training data for each specific classification task in a standard format; * The Deep NN Factory shall provide a standard representation of the trained NN so that the customer can run the network on his preferred execution infrastructure; * The Deep NN Factory shall be able to compress the delivered NN using a standard lossless or lossy compression technique before delivery; * The decompression of compressed NN shall be fast enough; * The Deep NN Factory shall provide a unique identification of the trained NN and support ways to identify an instance of a trained NN based on its configuration (e.g. taking into account topology, weights); * The Deep NN Factory shall be able to refine or retrain an existing NN provided by a customer in a standard format (compressed or not, even if the original NN was not pre-trained by it), together with additional data for the training if needed. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| I/O interfaces of the Factory are the main interfaces benefitting from a standard NN representation. The calling components may be either mobile devices or more complex systems. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| The update rate is not expected to be high, since the lifetime of a pre-trained classification model is expected to be relatively long. The expected size of each trained network (uncompressed) is measurable in several Gbytes. The expected bandwidth of the uplink (when transferring classification data for the training or refining) and of the downlink (when getting the resulting NN) is that appreciable from current state-of-the-art cloud services (several hundred Mbps) although this may be capped by the capacity of the mobile network when these operations are made in a mobile environment. The maximum acceptable latency is highly depending on the target application. For mobile applications, that need to be highly responsive to the user, the latency should be within few seconds. Other cases (such as content management applications), acceptable latency can be considerably higher (until few minutes). |

|  |
| --- |
| UC8A Personalized machine reading comprehension (MRC) application |
| MRC is an advanced deep learning technique in natural language processing. For the user’s question, it finds the answer in the natural language text using the neural network (NN). In order to answer user’s questions based on personal e-mail text and SMS messages, the MRC NN model should be learned from the personal questions and the personal texts. However, because of the privacy protection, it is difficult to collect and learn the personal data to send to the central cloud server. This use-case describes the case where the user’s edge device (i.e. smart phone) learns personalized MRC NN model using the personal MRC usage record. The personalized MRC NN model learned at the user’s edge device is sent to the central cloud server and used to build an updated version of shared MRC NN model. Fig. 1 shows an overall architecture for personalized MRC application.    **Fig.1 An overall architecture for personalized machine reading comprehension application** |
| **Overlap with other use cases** |
| Relation to UC10 (specific case of distributed training) |
| **Required features** |
| * The NN model transmitted to the device should include algorithm and parameter information for NN learning. * The NN model transmitted to the cloud server should include training metadata information such as number of training instances. * Significant size reduction of the neural network representation and trained weights. * (optional) The use of the network should be possible without inflating it to its original size in memory. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The NN building and distribution can be implemented in an interoperable way by different vendors of a backend system (e.g., operated by a broadcaster, service operator) and by client software for different devices.  This includes a platform independent compressed representation of a neural network, that can be used regardless of the platform and software frameworks used to implement personalized MRC. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Once the initial NN building is done, updates are expected to be infrequent, but potentially to a large number of devices. * The learning is performed when each individual device is in an unused and battery charging state. * Each device infrequently transmits an updated NN model to the cloud server. * The size of uncompressed NN model is about 500 and 1000 MB, depending on the model * For mobile devices, the download bandwidth may vary considerably due to the network type. * Latency is in most cases not critical, except for cases where a user downloads the mobile app to use it right away. |

|  |
| --- |
| UC8B Machine Translation application |
| In recent years, deep learning technique is applied in machine translation and shows a good performance enhancement. In speech translation, speech recognition and speech synthesis are used for processing input and output data and translation engine is used between speech recognition and speech synthesis modules. For all three modules, the neural network (NN) is applied successfully and provides good translation results. For a better translation results, new words and information on new adjacent words should be learned every time they were created and used. In fact, these new language resources of different languages are created and learned in distributed locations, sites, applications, devices or services. To apply new learned language resources to the translation system affects the translation accuracy and performances. This use-case describes the case where the on-line language resources such as new words and adjacency information are learned and applied to NN model incrementally. The learned translation NN model is sent to the central cloud server and used to build an updated version of shared translation NN model. Fig. 1 shows an overall architecture for translation application with incremental learning of new language information.  translation nn  **Fig.1 An overall architecture for machine translation application** |
| **Overlap with other use cases** |
| The existing use case of translation focuses on speech recognition and speech synthesis, but this new use case describes NN case applied in translation module itself. |
| **Required features** |
| * The NN model is transmitted and shared with incremental learning of new language resources * The NN model transmitted to the device should include algorithm and parameter information for NN learning. * The NN model transmitted to the cloud server should include training metadata information such as number of training instances. * Significant size reduction of the neural network representation and trained weights. * (optional) The use of the network should be possible without inflating it to its original size in memory. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The NN building and distribution can be implemented in an interoperable way by different vendors of a backend system (e.g., operated by a broadcaster, service operator) and by client software for different devices.  This includes a platform independent compressed representation of a neural network, that can be used regardless of the platform and software frameworks used to implement machine translation. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * After the initial NN building is done, updates are expected to be frequent to incorporate new knowledge on language resources collected on-line, and potentially to a large number of devices. * Each device frequently transmits an updated NN model to the cloud server. * The size of uncompressed NN model is about 500 and 1000 MB, depending on the model * For mobile devices, the download bandwidth may vary considerably due to the network type. * Latency is in most cases not critical, except for cases where a user downloads the mobile app to use it right away. |

|  |
| --- |
| UC10 Distributed training and evaluation of neural networks for media content analysis |
| This use-case concerns the distributed training and evaluation of neural networks for applications such as object detection and data generation performed on mobile devices and other devices. The open issues related to centralized solutions: 1) power in-efficiency for centralized training; 2) data communication; and 3) the deployment of the learned models.  It is good to have continuous training and evaluation capabilities, where the NN representation can be generated and shared in such distributed use cases. The validated model updates are shared with a central entity and/or other devices to improve the media analysis performance. Also, the training devices are assumed to have media capture capabilities, or other means for utilizing content for training.  In distributed (or federated) training, there is a central server which orchestrates the distributed training on several end devices (training devices) for the same (or similar) task. Each training device uses captured or other media to perform a local model update on the local version of the neural network. The estimated model update is not communicated to the central entity and/or to the other devices until it is validated. In fact, for each training-device performing a partial training, it will be extremely important to evaluate the *quality* and *integrity* of the partially-trained model, in order to decide whether the model-update estimated by a certain training-device should be taken into account (i.e., communicated to a central entity and/or to all other training devices) or be temporarily ignored. If it is successfully validated, the model update is then taken into account. This may mean that the model update is readily shared with all other training devices or it will be further combined with model updates from other training devices and then shared after a centralized validation.  Examples of aspects to be evaluated will be robustness to adversarial examples, performance based on the specific task (e.g., object detection, data-generation), etc.  The goal is to standardize the signaling content and format needed to evaluate neural networks. Signaling content may include data sent by a central entity to be input to the neural network, optionally also the ground-truth output (if the evaluation is done on the training device), feedback from the training device (neural network’s output if evaluation is done on a central entity, otherwise only the evaluation results), proposed weight updates in case of federated training.  To implement a \*model\* ensembling with distributed training to exploit the computational power of many distributed devices, in which training methods should be able to ensure the diversity among the modeal architectures and the training data across individual training devices.  Case A:  The server needs to signal, to different device, different model types & architectures & (optionally) initialization parameters & training datasets. This signaling is from server to each device in P2P mode.  Case B:  The server needs to signal, to different devices, the \*same\* model type & architecture & (optionally) initialization parameter & training datasets. This signaling is in broadcast mode.  The server sends indication to each devices signaling which connections‘ weights need to be masked (i.e. set to zero). This part of signaling is P2P as in case A.  Case C:  The server needs to signal, to different devices, the \*same\* model type & architecture & (optionally) initialization parameter & training datasets. This signaling is in broadcast mode.  The dropout is achieved by each device autonomously.  For all cases, devices send back the trained model, and meta-data (e.g. device identifer).  For some life-critical systems e.g. autonomous driving or medical diagonsis, the analysis of media contents has to be transparent and explaniable to human understanding. |
| **Overlap with other use cases** |
| Generalization of UC7&8, which describe specific cases of distributed training |
| **Required features** |
| * Validation of model updates in distributed NN from individual training devices * Communication of the whole (or partial) NN models between entities * The NN representation, for this use case, should support different types of neural networks, such as discriminative (for which input data is usually mapped to a classification or regression output), and generative (for which input data is transformed to data in the same domain as the input or to a different domain). Also, generative models may be conditional (on a specific input data) and unconditional (where the input is a sample from a latent space). * An efficient way to represent model updates. For example, if a device bandwidth is limited or a device did not receive a number of model updates for other reasons (e.g., offline), the device may request the server to send an aggregated model update which combines multiple update (or training) iterations, or the actual final model which were affected by the combined update. * Efficient compression schemes of NN models to be communicated * Evaluation of robustness to adversarial attacks * Detection of malicious model updates * Evaluation of interpretability of models (updates) e.g. by visualization. In case that NN compuatation is too complex to be human comprehensive, two features are needed: a) an objective measure of the interpretability of NN models is needed for the purpose of indication; b) more explainable NN modes are needed as surrogates to provide human comprehensive interpretations. * Time-stamping of the model updates. * To ensemble distributed models |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The interface which benefits from a standard neural network representation is the central entity which is interested in obtaining a high quality trained model.  In addition, in the case of federated training, another interface is the training devices themselves, as their partially-trained model will improve based on the weight updates sent by the central entity. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| Frequency   * In the case of distributed training, the neural network model updates are expected to be communicated in an adjustable manner (e.g., based on end device settings), and only when a device’s model update has been validated. * Also, at the server side, the model updates from multiple devices can be evaluated and combined and shared with the devices only if the performance improves sufficiently or only if the new model is evaluated as better with respect to other criteria. * In case of non-distributed training, the model updates are communicated only at the end of training (when a central entity only requires another device to perform the whole training).   Size   * From few KB to several hundreds of MB   Bandwidth   * It depends on the distribution channels, such as 2G, 3G, 4G, 5G mobile networks etc.   Maximum latency   * When the model update made by a certain device is validated by the central entity, the update is sent to the central entity and the maximum acceptable latency is application & implementation dependent which could be from ms to seconds or even longer.   After the central entity has combined all model updates received from several devices, it will send the combined weight update to all devices and the max acceptable latency is also application & implementation dependent. |

# Use cases on image/video processing and coding

|  |
| --- |
| UC11 Compact Descriptors for Video Analysis (CDVA) |
| MPEG CDVA specifies a compact descriptor for video segments, aiming to support search for specific objects across video collections. The standard describes the bitstream representation of such a descriptor, as well as those parts of the extraction process that are required to ensure interoperability of the resulting descriptors. Features extracted from deep convolutional neural networks (often named deep features) have been shown to be applicable for a wide range of computer vision tasks. CDVA makes use of features extracted from CNNs, which have been shown to provide good performance and are complementary to traditional features such as those included in CDVS.  There are two levels of conformance: With strict conformance, the neural network is exactly defined, and thus the issue of transmitting the network is only one of the initial deployment of the application. With loose conformance the network may be exchanged, but the same network representation needs to be shared between all parties to establish interoperability. In this case, an update of the neural network may need to be updated more frequently.  Intended use cases of CDVA include descriptor extraction on devices such as smart phones or set top boxes. |
| **Overlap with other use cases** |
| Related to UC13 |
| **Required features** |
| * Significant size reduction of the neural network representation and trained weights. * If lossy compression is used, the performance of using a compressed representation of the network must be similar to that of using an uncompressed network. * (optional) The use of the network should be possible without inflating it to its original size in memory. * (optional) Matching descriptors extracted using a compressed network with descriptors extracted using an uncompressed network results in only small performance loss compared to matching two descriptors extracted using uncompressed networks. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The aim of CDVA is to decouple descriptor extraction and matching/retrieval, so that these two steps can be implemented in an interoperable way by different vendors of a backend system (e.g., operated by a broadcaster, service operator) and by client software for different devices.  This includes a platform independent compressed representation of a neural network, that can be used regardless of the platform and software frameworks used to implement CDVA. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Updates are expected to be infrequent (maybe a few times per year), but potentially to a large number of devices. * The size of the uncompressed network is about 500-600 MB. * For mobile devices, the download bandwidth may vary considerably due to the network type. For set top boxes this would have to be done with as part of a regular software update, thus the size is an issue. * Latency is in most cases not critical, except for cases where a user downloads the mobile app to use it right away. |

|  |
| --- |
| UC12 Image/Video Compression |
| Recently, studies on still image and video compression based on deep neural network (DNN) have been actively conducted, and it has been reported that its performance can be comparable to the conventional image compression standard[[4]](#footnote-4). Image/video compression using DNN is widely thought to be a new compression method. RNN (Recurrent Neural Network) is mainly applied for image/video compression. In particular, for video compression, different approaches have been attempted to apply deep learning techniques on a tool-by-tool basis or on the entire codec.  DNN based image/video compression requires coded representation of neural network in the following two aspects. Image/video encoders and decoders are used at different locations in general applications codec. In other words, a feature vector, which is the output of the encoder to the input image/video, is stored or transmitted, and the original input image/video is reconstructed at the decoder. Therefore, the coded representation of the trained neural network of a codec is needed to be delivered to decoders. In addition, re-trained network is desirable to be updated periodically.  In addition, NNR may be needed in a specific application environment of the image/video codec. For example, there may be available different NNs customized to categories of image/video to be encoded. In this case, the coded representation of NN associated with a given input image/video may be required to support such application. |
| **Overlap with other use cases** |
| Related to UC11 and UC13 (using specific types of feature extraction/processing) |
| **Required features** |
| * High efficient representation of neural network including network model and trained weights * Low complex decoding (computation/memory) of the compressed representation of neural network * High error robustness for the coded representation of neural network |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| * DNN based image/video codec can be used in an interoperable way in diverse media services (broadcasting, mobile streaming, visual communications, etc.). * Implementation of a DNN based decoder can be supported in an interoperable way by different vendors * Who: service providers (broadcaster, IPTV, mobile streaming, etc.), network operators (mobile), vendors |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| * Once a DNN based decoder is initialized, updates of NN are expected to be infrequent (periodic update may be feasible depending on applications, for example, once a month), but potentially to a large number of devices. * The estimated size/distribution bandwidth of the presentation of DNN encoder/decoder may not be critical to be initialized infrequently. * For the representation of output of an encoder, the size/bandwidth should be smaller than that of the conventional codec. * After initialization of a decoder, latency should be comparable to the current latency in the applications of broadcasting, communication, and streaming. |

In DNN based approaches on image/video compression, deep learning techniques can be applied to either entire codec or parts of codec. Accordingly, use cases and/or requirements for each case in details may be different although most of them are common, and we present the analysis of use cases and requirements of both cases in this section.

## Tool-by-tool case (UC12-A)

The existing works on a tool-by-tool basis typically apply deep learning to intra prediction, intra mode decision, inter prediction, in-loop filtering [3]-[6], etc. In these works, DNN techniques are used in the generation of the prediction block for intra prediction [3], filtering for in-loop filtering [4], classification for intra mode decision [5], and so on. All the works give better performance in terms of coding efficiency, but the complexity is still excessively high compared to the existing conventional methods. Therefore, such complexity issue should be addressed in the coded representation of neural networks. For instance, model compression can play an important role in the practical application of DNN techniques. In addition, the network structure trained in a training framework should be transmitted to both of encoder and decoder, or one of them only.

### Scenario of use case



Fig. 1 Overall configuration of a tool-by-tool case.

Fig. 1 shows a system configuration of DNN based codec on a tool-by-tool basis. Considerations on the use case and its scenario are given below.

* Network training framework, user – It is assumed that a set of DNN based coding tools are designed and trained by a network training framework in advanced. One or more coding tools are selected and transmitted to an inference engine of image/video codec in a given application. Furthermore, multiple tools can be available to support the same functionality of codec. For an example, more appropriate tool with higher performance can be selected according to the type of input content to be compressed. These configurations are prepared and set by a user who manage the network training framework.
* DNN based coding tool – In the image/video compression, some coding tools are required in both encoder and decoder, and some other tools are used encoder or decoder only. For example, in-loop filtering should be done in both encoder and decoder. On the other hand, intra mode decision is done in an encoder, and then the determined prediction mode is signalled to a decoder. Therefore, two types of DNN based coding tools are need to be considered, and the type should be identified: Type-A tools should be transmitted to encoder and decoder together; Type-B tools are need to be transmitted to encoder or decoder only.
* Representation of NN – The results of trained DNN based tools, namely trained networks, should be delivered and exchanged in an interoperable manner. This is achieved by specifying an interoperable format for representing trained networks including network structure and trained weights. The interoperable format need to represent high-level information regarding to system configuration as well as trained networks. For example, in the use case on a tool-by-tool basis, the type of tool should be represented by the interoperable format to indicate whether the tool is applied to both encoder and decoder, or encoder or decoder only.
* Compression of NN – The trained networks are delivered to an accelerator library in which the trained networks are compressed and then represented in an interoperable format. Compression of the neural network using the acceleration library can be optional. In other words, when the trained networks may be compact enough to be delivered to inference engines without compression, or when the compression significantly degrades the performance of codec, the trained networks can be directly transmitted to the inference engine without compression.
* Inference engine – In the inference engine of image/video encoder, the coding tool represented as an interoperable format is used on a tool-by-tool basis. By the way, the encoder can adaptively select either the DNN-based coding tool or the existing conventional coding tool based on the RD-cost rather than replacing the conventional method for enhancing coding performance.
* Output bitstream of encoder – The output bitstream of an image/video encoder is transmitted to a decoder. In this step, DNN based coding tools are used in both inference engine or one of them. It is assumed that the compressed bitstream is basically compatible to the existing image/video coding standards even if parts of encoding and/or decoding are done by the DNN based coding tools.

## End-to-end case (UC12-B)

Unlike a tool-by-tool basis, an entire basis applies deep learning techniques to end-to-end processing of encoder and decoder. In other words, the whole processing of encoding and decoding is done by DNN. Multiple DNN models can be available for image/video compression. While the tool-by-tool case need to identify which coding functional block is supported by which DNN based coding tool, the end-to-end case only need to identify which model of DNN is available. Due to mainly the complexity, an entire codec basis has been applied to image compression only so far [7].



Fig. 2. Autoencoder for image/video compression on an codec entire basis

An autoencoder neural network is basically used to compress image/video compression on an entire codec basis. As shown in Fig. 2, an autoencoder consists of encoder and decoder both of which are neural networks. The encoder outputs a feature vector that reduces the dimension of the input image, and the decoder reconstruct the input image from the input feature vector. The feature vector, which is the output of the encoder, is represented and delivered as compressed bitstream through quantization and entropy coding. This means that both encoder and decoder are trained, and then the trained encoder networks is delivered to an inference engine of encoder, and the trained decoder network is delivered to the decoder.

### Scenario of use case

### 

### Figure 3. Overall configuration of an entire codec case

In a network training framework, one or more DNNs are trained and the trained networks are represented as an interoperable format. When multiple trained DNNs are available, an appropriate one for a given application is selected by a user, and downloaded to an inference engine. The represented trained network is compressed in an accelerator library, then delivered to an inference engine of codec, or directly delivered to an inference engine without passing the accelerator library in some cases as explained in the section 2.1.

In an entire codec basis, a feature vector that is the output of the encoder to the input image/video is transmitted to a decoder in which the original input image/video is reconstructed. The compression of the feature vector, which is not supported in the existing image/video compression standards, is also crucial in addition to the coded representation of the trained networks. However, this issue may be a separate issue regarding to the standard of DNN based image/video coding.

|  |
| --- |
| UC13 Distribution of neural networks for content processing |
| Recently, analysis and processing of media such as images and videos require the application of neural networks.  One example of content processing is applying neural network based super resolution (NSR) on a video at the client side, so that the video can be transmitted to that client at reduced resolution and thus save bandwidth.  The neural network performing NSR may be trained to increase the resolution by a specific factor (e.g., 2), thus the sender needs to down-sample the content by the same factor and signal to the client the neural network to be used. This may consist of either signaling the neural network type (e.g., the super resolution factor) if the client already has that neural network or the actual neural network weights and topology.  Furthermore, the neural net may be specifically trained (or fine-tuned) on the content which is being transmitted, for example on a video. In this case, the neural net needs to be transmitted to the client either before the video transmission, during the video transmission in case different networks are trained for different temporal portions of the video. This can be extended to other content processing techniques apart from super resolution, such as other content enhancing neural networks. In this use case proposal, we use neural network and model interchangeably. |
| **Overlap with other use cases** |
| Relation to UC11 and UC12 |
| **Required features** |
| * Signaling is required between server and client in order for client and server to negotiate the relevant models or model update (topologies/architectures, and associated weights) to be used with the content. * If the model is not already present at client side, efficient representation and transmission of the model from server to client which performs the content processing. * If models are fine-tuned on the specific content, the server is required to transmit the model to be applied to the whole content. * If models are fine-tuned on different temporal portions of the content, the server is required to transmit the model to be applied to a certain temporal portion and the content in that temporal portion.   The server is required to **send** low quality content data to devices, together with NN fine-tuned on that content.  For case A: NN is fine-tuned for the whole content. The server **signals/sends** NN only once, either before or during the streaming of content. The contents and the associated NN should be **paired** e.g. by unique ID.  For case B: Different NNs are optimized for different portions of contents (either split temporally or otherwise). The server signals each NN or differences before or during the streaming of each portion of contents.  Again, the **paring** of different portions of contents and associated NN is needed. |
| **Which is/are the interface(s) that benefit from a standard neural network representation? Who is in control of the components (e.g., mobile devices, embedded processors) involved?** |
| The content retrieval and negotiation interfaces will benefit from a standardized signaling of neural network representations, from inter-operability perspective.  Moreover, the media consuming devices will benefit from a standardized neural network representation which is suitable to their AI capabilities. |
| **How often are neural networks expected to be transmitted/updated in this use case (per unit, in total)?**  **What is the estimated size of these neural networks?**  **What is the expected bandwidth of the distribution channel?**  **What is the maximum latency that is acceptable?** |
| Frequency   * In the case of a single model for all contents, the model needs to be transmitted only once until a new version of the neural network is available. E.g., every 1 month. * In case the model is fine-tuned on each specific content, the model needs to be transmitted with every new content, so even multiple times a day. * In case the model is fine-tuned on each temporal portion of a certain content, the model needs to be transmitted at every content portion. E.g., for a video, every scene change. For example, every 1-5 minutes.   Size   * Sizes can range from few KB to several hundreds of MB.   Bandwidth   * The bandwidth may depend on the specific distribution channel. One example is 2G, 3G and 4G networks.   Acceptable latency   * In case of a single model for all contents, any small latency is acceptable, even in the order of seconds or minutes. * In case the model is fine-tuned on each content, maximum acceptable latency is about one second * In case the model is fine-tuned on each temporal portion of content, the maximum acceptable latency is very small, e.g., a fraction of a sampling period. For example, for a video streaming, latency may be 3 milliseconds (one tenth of the sampling period). |

# Summary of Requirements

This table summarises the different requirements described, and the use cases that make reference to them.

|  | **distribution** | | | | | | | | | **(re)training** | | | **processing** | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **9** | **14** | **15** | **7** | **8** | **10** | **11** | **12A** | **12B** | **13** |
| **Compressed representation** |  | | | | | | | | | | | | | | | |
| Exchange representation | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Efficient representation of the network | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Support lossless compression |  | X | X |  |  |  |  | X |  | X |  |  |  |  |  |  |
| Support lossy compression | X | X | X |  |  |  |  | X |  | X |  |  | X |  |  |  |
| Comparable performance of compressed network than original network | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Scalable compression | X |  |  |  |  |  |  | X | X |  |  |  | X |  |  |  |
| Inference with compressed network |  | X | X |  | X |  |  |  |  |  | X |  | X |  |  |  |
| Must support compression methods that do not require access to the original training data, may support others in addition | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Represent partial NN models | X |  | X | X | X | X | X |  | X | X | X | X |  |  |  | X |
| enable efficient incremental representation of NNs[[5]](#footnote-5) | X |  | X | X | X | X | X |  | X | X | X | X |  |  |  | X |
| Low computational complexity decoding |  | X |  | X | X | X |  | X | X | X | X |  | X | X | X |  |
| Low memory consumption |  | X |  | X | X | X |  | X |  |  | X |  |  | X | X |  |
| Support representation of different types of artificial neural networks |  |  |  |  |  |  |  |  |  |  | X | X |  | X | X |  |
| **Metadata** |  | | | | | | | | | | | | | | | |
| Unique identification and time-stamping of an instance of a trained NN | X |  | X | X | X |  |  | X | X | X | X | X | X |  |  |  |
| description of training metadata and reference to training data |  |  |  |  |  |  |  |  |  | X | X | X |  | X | X | X |
| Support validation of model updates | X |  |  |  |  | X |  |  |  |  |  | X |  |  |  | X |
| Metadata to support interpretability of trained models |  |  |  |  |  |  |  |  |  |  |  | X |  | X | X |  |
| Metadata on required capabilities of inference engine | X |  | X |  | X | X |  |  | X |  |  | X | X | X | X | X |
| Extension mechanism to support application specific metadata |  |  |  |  |  |  |  |  |  |  |  | X |  | X | X | X |
| **Robustness and security** |  | | | | | | | | | | | | | | | |
| Error resilience of compressed representation | X |  |  | X |  |  |  | X |  |  |  | X |  | X | X | X |
| Ability to detect manipulation of the compressed network representation[[6]](#footnote-6) |  |  |  | X |  |  |  |  |  |  |  | X |  |  |  |  |
| **Other[[7]](#footnote-7)** |  | | | | | | | | | | | | | | | |
| Interoperability of components using compressed and uncompressed versions of the network[[8]](#footnote-8) |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |  |
| Signaling to negotiate relevant model updates |  |  |  |  |  | X |  |  |  |  |  | X |  |  |  | X |
| Mechanism to guide execution of NN models |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  | X |
| Interface for providing training data |  |  |  |  |  |  |  |  |  | X | X | X |  | X | X | X |
| Signaling for negotiating the re-use of NN models |  |  |  |  |  |  | X |  |  |  |  | X |  |  |  | X |
| Future proof w.r.t delivery modes |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| Robustness to adversarial attacks |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |

1. http://www.zdnet.com/article/huawei-mate-10-pro-camera-enhancing-auto-mode-through-artificial-intelligence/ [↑](#footnote-ref-1)
2. https://play.google.com/store/apps/details?id=com.microsoft.translator&hl=en [↑](#footnote-ref-2)
3. |  |
   | --- |
   | For example, over 400 million surveillance cameras will be installed in China by 2020,  http://www.bbc.com/news/av/world-asia-china-42248056/in-your-face-china-s-all-seeing-state |

   [↑](#footnote-ref-3)
4. G. Todericiet et al., “Full Resolution Image Compression with Recurrent Neural Networks,” In Proc. CVPR 2017, July 2017.  
   C. Kin, B. Coker, “Video Compression Using Recurrent Convolutional Neural Networks,” [↑](#footnote-ref-4)
5. for inter-model prediction or incremental updates [↑](#footnote-ref-5)
6. The representation shall support any category of network, e.g. feedforward networks such as CNN and autoencoder, recurrent networks such as LSTM, etc. [↑](#footnote-ref-6)
7. To be further discussed which of these requirements are considered in scope. [↑](#footnote-ref-7)
8. Only relevant when using the NN for feature extraction or content extraction [↑](#footnote-ref-8)