**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11/N17929**

**October 2018, Macau SAR, CN**

**Title: Evaluation Framework for Compressed Representation of Neural Networks**

**Source Requirements Subgroup**

**Status: Approved**

# Introduction

There is an increasing demand for efficiently deploying trained deep learning models. The MPEG on Compressed Representation of Neural Networks aims to define an efficiently coded, interpretable and interoperable representation for trained neural networks.. This document describes a draft evaluation framework for technology for compressed representation of neural networks.

# Evaluation Framework

The straightforward evaluation approach is to evaluate the compression ratio of compressed model and the performance with the reconstructed model. The proposed evaluation framework is shown in Figure 1. Firstly, the original model performance along with the model size is recorded. Then the model is compressed using the method under test (which could optionally include retraining) and the compressed model size is evaluated. After decompression, the reconstructed model is used in the target application and the model performance is evaluated.



Figure 1: Overview of the evaluation process.

For lossless compression methods, the performance of the reconstructed model does not need to be evaluated, as it is expected to be identical to the original model. Only the correct reconstruction of the model has to be checked against the original model.

For lossy compression, the evaluation of the performance of the reconstructed model can only be measured in a particular application. The evaluation framework thus includes a list of specific applications, in which the compressed representation is evaluated. For each of these applications, one more specific performance metrics are defined. A preliminary list of selected applications and the evaluation procedures for these applications are provided in Section 4.

# Evaluation Metrics

As the evaluation framework aims to evaluate the compression rate and model performance, several evaluation metrics shown in Figure 1 are listed below.

## Model size (O\_size, R\_size) (for lossy compression only)

During transmission/storage, the compressed model size may be much smaller than the original model size O\_size, and the reconstructed model size R\_size should also be considered as the reconstructed model size indicates how much data were lost. Therefore, we also compare the R\_size with original model size O\_size.

The model sizes O\_size and R\_size are measured as the sum of the size of all raw parameters/weights of the serialized model.

## Compressed model size (C\_size)

One of the most important evaluation criteria is to compare the compressed model size between different approaches. During transmission/storage, the compressed model size may be much smaller than the original model size O\_size. The size of the compressed model C\_size is measured as the sum of the size of all compressed/encoded parameters/weights of the serialized model.

## Reconstructed model performance (O\_Per, R\_Per) (for lossy compression only)

Here we compare the model before and after compression, denoted as O\_Per and R\_Per. In other words, the O\_Per and the R\_Per should be as close as possible.

For a set of use cases (see Section 4), specific performance metrics for each of these have been defined. The evaluation process and metrics for selected use cases are described in Section 4.

## Reconstructed model performance including retraining (O\_Per, RR\_Per) (for lossy compression only)

Optionally, retraining may be applied to refine a compressed neural network. We measure the performance of using the reconstructed model RR\_Per after performing compression and retraining.

For a set of applications (based on the identified used cases), specific performance metrics for each of these will be defined. The evaluation process and metrics for selected use cases are described in Section 4.

## Runtime and memory complexity

In addition, runtime and memory complexity of the compression and decompression step will be measured.

The following metrics shall be reported

* Runtime of compression of the model, and runtime of retraining (if applicable)
* Memory consumption of the model compression step
* Runtime of decompressing/loading the model
* Memory consumption of the model decompression step

For the CfE, proponents are required to report the hardware and software configuration used for running the runtime measurements. For the CfP, it is intended to define a reference configuration.

It is recommended to repeat runs and reports averaged runtimes in order to get reliable runtime measurements.

## Incremental representation (C\_ratio)

NOTE: This section does not apply to the CfP closing in March 2019.

For the incremental representation of NNs, the compression ratio of the model update vs. the complete model will be measured. This applies to cases where the update is encoded referring to a single model as well as when it is encoded referring to a set of models (e.g., a model can beincrementally based on an existing model set, which has great similarity wrt. models/layers/weights reusing).

The performance will be measured with the reconstructed model after applying the incremental update using the metrics for reconstructed models described above.

In addition, the runtime and memory consumption of applying the incremental update of the model shall be measured.

# Measuring Model Performance in Specific Use Cases

This section describes the performance measurement process and metrics for a set of selected use cases.

## Visual object classification (UC2 Camera app with object recognition, UC4 Large-scale public surveillance)

For the use cases including visual object classification, a set of different NN models trained on ImageNet [5] is used.

### NN Models

The following models are provided:

|  |  |  |
| --- | --- | --- |
| Model name | Framework | URLs |
| VGG16 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/VGG16https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| GoogLeNet(Inception) V3 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/InceptionV3https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| ResNet50 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/ResNet50https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| MobileNet | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/MobileNethttps://github.com/fchollet/deep-learning-models/releases/download/v0.6/mobilenet\_1\_0\_224\_tf.h5 |
| NASNet | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/NASNetLargehttps://github.com/fchollet/deep-learning-models/releases/download/v0.8/NASNet-large.h5 |
| DenseNet169 | Tensorflow | https://github.com/pudae/tensorflow-densenet |
| DenseNet201 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/DenseNet201https://github.com/fchollet/deep-learning-models/releases/download/v0.8/densenet201\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| DenseNet201 | PyTorch | https://github.com/pytorch/visionhttps://download.pytorch.org/models/densenet201-c1103571.pth |
| DenseNet201 | Torch | https://drive.google.com/open?id=0B8ReS-sYUS-HaDdpNmlWRjJkd3c |
| Xception | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/Xceptionhttps://github.com/fchollet/deep-learning-models/releases/download/v0.4/xception\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| AlexNet | PyTorch | https://github.com/pytorch/examples/tree/master/imagenethttps://download.pytorch.org/models/alexnet-owt-4df8aa71.pth |

### Test framework and data

As test framework and dataset the Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset [4] which is subset of the ImageNet [5] evaluation procedure and data set.

In addition, models trained on CIFAR10 and CIFAR100 (PyTorch, https://github.com/Eric-mingjie/network-slimming/tree/master/models) may be used as an easier starting point, and the CIFAR evaluation procedure and data set [10] may be optionally provided. Nonetheless, the submission of results for ImageNet is required.

### Evaluation metrics

Top-1 and Top-5 classification performance shall be reported.

## UC11 Compact Descriptors for Video Analysis (CDVA)

In the MPEG Compact Descriptors for Video Analysis (CDVA) specification [8], deep features are extracted via standardized deep model, which utilizes VGG-16 trained on ImageNet as the extractor model. However, other models such as Alexnet, Resnet50 are also able to perform as the deep feature extractor, and CDVA allows the use of custom NNs.

### NN Models

The following trained models are available:

1. VGG-16
2. Alexnet
3. Resnet50

The models can be downloaded from https://drive.google.com/open?id=13L4\_T5p40sUF5HpY077TkMxcn4qUvOXZ

### Test framework and data

As test framework, the CDVA Test Model CXM5.1 [6] or later shall be used. The data set to be used for evaluation is the CDVA data set described in [7].

### Evaluation metrics

The pairwise matching experiment as described in [7] shall be performed for the 16K working point. The True Positive Rate at 1% False Positive Rate shall be reported as metric.

## UC12A Image/Video Compression – Tool-by-tool use case

For this use case two representations of neural networks are provided, each of which is used to replace the in-loop filter in the JEM reference software codec and the VTM reference software codec, respectively.

### NN Models

**Filter for JEM**

For the filter in the JEM, the model file format of the framework itself is used to represent a trained network in Caffe.

The trained model can be downloaded from https://jvet.hhi.fraunhofer.de/svn/svn\_HMJEMSoftware/branches/candidates/HM-16.6-JEM-7.1-AHG9-I0022/bin/

**Filter for VTM**

For the filter in the VTM, the model trained in Keras is represented in the forms of JSON and HDF5. In other words, the model structure is represented by JSON library, and the weights are stored in the HDF5 file for loading and testing.

The trained model is available at https://drive.google.com/open?id=13RbDKII0PTNFxIEdl5vxfTWauAAuY8Ns

### Test framework and data

For both filters, the JVET CTC [1] shall be used as the test data set, testing with the QPs defined in the CTC.

**Filter for JEM**

For the JEM, a modified version using the NN is available at <https://jvet.hhi.fraunhofer.de/svn/svn_HMJEMSoftware/branches/candidates/HM-16.6-JEM-7.1-AHG9-I0022/>

Build instructions can be found in the README file in this directory.

This subsection shows part of the software contents of JVET-I0022 that proposes a CNN-based in-loop filter in JEM [2].

Information required for training and testing of the neural network for in-loop filter in JEM [2] is presented below.

* Required libraries of Caffe for interpreting an input file of “prototxt” or “caffemodel” in the test step
* Configuration for encoding/decoding of JEM with the proposed CNN-based in-loop filter

|  |
| --- |
| * AdditionalDependencies: // libraries to be required for running the neural network
	+ Model file usage
		- libprotobuf.lib
		- libopenblas.dll.a
	+ GPU Optional
		- cuda.lib
		- curand.lib
		- cudnn.lib
* AdditionalIncludeDirectories: // header file directory
	+ (Caffe\_PATH)\NugetPackages\boost.1.59.0.0\lib\native\include;
	+ (Caffe\_PATH)\NugetPackages\gflags.2.1.2.1\build\native\include;
	+ (Caffe\_PATH)\NugetPackages\glog.0.3.3.0\build\native\include;
	+ (Caffe\_PATH)\NugetPackages\protobuf-v120.2.6.1\build\native\include;
	+ (Caffe\_PATH)\NugetPackages\OpenBLAS.0.2.14.1\lib\native\include;
	+ (Caffe\_PATH)\caffe-windows\include;
* GPU Optional:
	+ (Cuda\_PATH)\NVIDIA GPU Computing Toolkit\CUDA\v8.0\include;
* Environment path:
	+ (Caffe\_PATH)\caffe-windows\Build\$(Platform)\$(Configuration);
 |
| **Here is the extra JVET-I0022 configurations compared with JEM7.0:*** For Encoder:
	+ UseGPU: // Use GPU or not, the default setting is 0, the command format is '-gpu';
	+ GPUID: // The ID of the GPU you choose to use, the default setting is 0. The command format is '-gpuid';
	+ CNNFilterModelFileI: // The path of the file including the network structure, which is with postfix ".prototxt". The command format is '-cnnfmi'
	+ CNNFilterTrainFileI: // The path of the file including the trained parameters, which is with postfix ".caffemodel". The command format is '-cnnfti'
* For Decoder, you need to set the the CNN files in your scripts:
	+ -cnnfmi: // The path of the file including the network structure, which is with postfix ".prototxt";
	+ -cnnfti: // The path of the file including the trained parameters, which is with postfix ".caffemodel";
	+ -gpu: // Use GPU or not, the default setting is 0
	+ -gpuid: // The ID of the GPU you choose to use, the default setting is 0
* Our CNN model and prototxt files were stored under the folder of bin, we also give the scripts and configurations in the folder of bin and cfg for demonstration.
 |
| * (Caffe).protxt file includes only neural network structure
* (Caffe).caffemodel file include both trained weights and structure
 |

**Filter for VTM**

For the filter in the VTM, the standard version of the VTM shall be used, and the filter is applied as a post-processing step using a separate tool. For getting access to this tool, please contact hcmoon@kau.kr

The subsection describes the details on a network model of CNN for replacing in-loop filter in VTM 1.0 [3]. In other words, an example of the representation of the trained network structure and network parameters (weights) is presented.

Parts of source code to define a CNN for in-loop filter and to save/load the trained network in Keras are shown below.

|  |  |
| --- | --- |
| Specification of model structure in Keras | Saving/loading of the trained network in Keras |
| from keras.models import Modeldef build\_model(input\_size): inputs = Input(shape=(input\_size[0], input\_size[1], 3)) conv\_1 = Conv2D(64, (9, 9), strides=(1, 1), padding='same')(inputs) act\_1 = Activation('relu')(conv\_1) conv\_2 = Conv2D(32, (3, 3), strides=(1, 1), padding='same')(act\_1) act\_2 = Activation('relu')(conv\_2) conv\_22 = Conv2D(32, (5, 5), strides=(1, 1), padding='same')(act\_1) act\_22 = Activation('relu')(conv\_22) conv\_23 = Conv2D(32, (7, 7), strides=(1, 1), padding='same')(act\_1) act\_23 = Activation('relu')(conv\_23) merge\_1 = concatenate([act\_2, act\_22,act\_23], axis=3) deconv\_1 = Conv2D(3, (5, 5), strides=(1, 1), padding='same')(merge\_1) act\_3 = (deconv\_1) model = Model(inputs=[inputs], outputs=[act\_3])  return model | from keras.models import model\_from\_jsondef save\_model(model, name):  json = model.to\_json() with open(name, 'w') as f: f.write(json)def load\_model(name): with open(name) as f: json = f.read() model = model\_from\_json(json)return modelmodel.save\_weights('weights.hdf5')model = load\_model('model.json')model.load\_weights('weights.hdf5') |

### Evaluation metrics

The following metrics shall be applied:

* BD-rate
* PSNR

measured between the original and the reconstructed frame, for both the filter with the original and the compressed network.

## UC12B Image Compression – End-to-end use case

It is an example of applying neural network to image compression in an end-to-end approach and generally has autoencoder structure.

### NN Models

Parts of code to define a CNN for image compression with auto-encoder and to save/load the trained network in Tensorflow are shown below. The model trained in Tensorflow is represented in the forms of HDF5 instead of the model file format. The HDF5 file also includes the trained weights of the model and training configuration (such as loss function, optimizer, etc.).

|  |  |
| --- | --- |
| Specification of model structure in Tensorflow | Saving/loading of the trained network in Keras |
|  **# encoder**encoder\_model.add(Convolution2D(16, 3, strides=1,padding='same') encoder\_model.add(Convolution2D( 16, 3, strides=1, padding='same')encoder\_model.add(Activation("relu")) encoder\_model.add(MaxPool ing2D(pool\_size=2, strides=2, padding='same')) encoder\_model.add(MaxPool ing2D(pool\_size=2, strides=2, padding='same')) # **decoder** decoder\_model.add(UpSampling2D((2, 2), data\_format='channels\_last')) decoder\_model.add(UpSampling2D((2, 2), data\_format='channels\_last')) decoder\_model.add(Convolution2D(16, 3, strides=1, padding='same')) decoder\_model.add(Convolution2D(16, 3, strides=1, padding='same'))self.encoder = encoder\_modelself.decoder = decoder\_model | encoder.save\_weights('encoder\_weights.h5')decoder.save\_weights('decoder\_weights.h5')encoder.load\_weights('encoder\_weights.h5')decoder.load\_weights(‘decoder\_weights.h5’) |

The trained model is available at https://drive.google.com/open?id=13RbDKII0PTNFxIEdl5vxfTWauAAuY8Ns

### Test framework and data

As test data, CIFAR100 [10] shall be used.

The test conditions are listed in the following table.

|  |  |
| --- | --- |
| Test data | CIFAR 100 |
| Image size | 32 x 32 x 3 |
| Color component | RGB |

### Evaluation metrics

As evaluation metrics PSNR and SSIM shall be used, comparing with the original image.

## Audio classification (UC16A Acoustic Scene Classification, UC16B Sound Event Detection)

### NN Models

The models and data described within this document include trained neural network models used for audio classification (i.e. more precisely they are used for music genres classification, and instrumental/vocal detection).

The following script includes the access to the training data, training of CNN model, and computation of classification accuracy: https://github.com/slychief/ismir2018\_tutorial/blob/master/Part\_1\_Convolutional\_Neural\_Networks.ipynb

|  |  |
| --- | --- |
| Network Architecture | Convolutional Neural Network |
| Model Format | Keras .h5 file |
| Music genre classification model | https://github.com/slychief/ismir2018\_tutorial/blob/master/models/part\_1\_cnn\_genre.h5 |
| Instrumental/vocal classification models | https://github.com/slychief/ismir2018\_tutorial/blob/master/models/part\_1\_cnn\_inst\_vocal.h5 https://github.com/slychief/ismir2018\_tutorial/blob/master/models/part\_1\_cnn\_inst\_vocal\_2.h5 |

### Test framework and data

Input data: Mel spectograms available for download at

https://owncloud.tuwien.ac.at/index.php/s/vTO0vD6L1iGGpul

Ground truth data (music Genres labels):

https://github.com/slychief/ismir2018\_tutorial/blob/master/metadata/ismir2018\_tut\_part\_1\_genres\_labels\_full.csv

Ground truth data (instrumental/vocal classification):

https://github.com/slychief/ismir2018\_tutorial/blob/master/metadata/ismir2018\_tut\_part\_1\_instrumental\_labels\_full.csv

### Evaluation metrics

For audio classification: classification accuracy.

For sound event detection: segment-based error rate.

The segment-based error rate is defined as [9]



# References

1. A. Segall, et al, “JVET common test conditions and evaluation procedures for HDR/WCG Video Coding,” JVET document, JVET-D1020, Oct. 2016.
2. L. Zhou, et al, “Convolutional Neural Network Filter (CNNF) for intra frame”, JVET document, JVET-I0022, Jan. 2018.
3. H. C. Moon, J.-G. Kim, “CNN Based In-Loop Filter for Versatile Video Coding (VVC),” The Korean Institute of Broadcast and Media Engineers Conference, Jeju, Korea, pp. 270-271, 2018.
4. Large Scale Visual Recognition Challenge 2012 (ILSVRC2012), http://image-net.org/challenges/LSVRC/2012/browse-synsets.
5. ImageNet, http://www.image-net.org/.
6. CDVA CXM5.1 <https://s3-eu-west-1.amazonaws.com/mpegcdva/mpeg-cxm5.1.zip>
7. N15338, Evaluation Framework for Compact Descriptors for Video Analysis - Search and Retrieval, Warsaw, Poland, July 2015
8. N17600, Text of ISO/IEC DIS 15938-15 Compact Descriptors for Video Analysis, San Diego, CA, USA, Apr. 2018.
9. K. Feroze, A. R. Maud, "Comparison of Baseline System with Perceptual Linear Predictive Feature Using Neural Network for Sound Event Detection in Real Life Audio," Detection and Classification of Acoustic Scenes and Events 2017, <http://www.cs.tut.fi/sgn/arg/dcase2017/documents/challenge_technical_reports/DCASE2017_Feroze_179.pdf>
10. Available at [Online]: <https://www.cs.toronto.edu/~kriz/cifar.html>