

**ISO/IEC JTC 1/SC 29/WG 11 N 18575**

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**Document type: Approved WG 11 document**

**Title: Evaluation Framework for Compression of neural networks for multimedia content description and analysis**

**Status: Approved**

**Date of document: 2019-07-19**

**Source: Video**

**Expected action:**

**No. of pages: 15**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: mpeg.chiariglione.org**

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11/N18575**

**July 2019, Gothenburg, SE**

**Title: Evaluation Framework for Compression of neural networks for multimedia content description and analysis**

**Source Video Subgroup**

**Status: Approved**

# Introduction

This document describes the evaluation framework for MPEG Compression of neural networks for multimedia content description and analysis.

# 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.

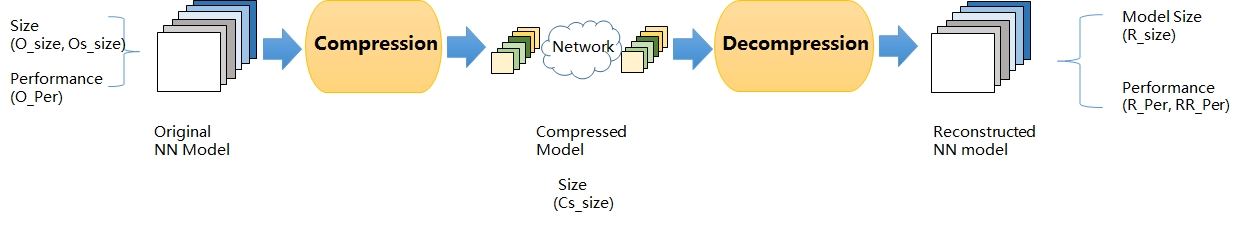


Figure 1: Overview of the evaluation process.

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.

The procedures for measuring runtime, the number of operations and energy consumption are described in Section 5.

For reporting results, the attached evaluation sheet shall be used.

# Test conditions

The following test conditions are defined.

## Availability of data

For each of the tasks, a small part of the original training data is reserved ("parameter tuning set") for (hyper)parameter search. As a starting point, models trained on the entire original training data set are used. For any data-dependent transformations requiring larger amounts of data, the training data set excluding the parameter tuning set shall be used.

For methods that include operations that use external data in optional steps, results for the following conditions shall be reported:

* without using any data: no use of data other than that of the input neural network is permitted
* using only parameter tuning data (without labels): Only data from the parameter tuning set defined for each of the applications may be used, but without labels/annotation, i.e., only that data that is input to the neural network shall be used, but no information related to the expected output.
* using training data: The original training data set excluding the parameter tuning set may be used, while the parameter tuning set may only be used for (hyper)parameter search.

## Hyperparameters for compression method

Hyperparameters of the compression method are a small set of parameters chosen to control the trade-off between compression ratio and performance in the specific task, and typically determine (together with the input data) the full set of parameters of the method.

* For each working point, a set of hyperparameters shall be chosen. Results shall be reported for these hyperparameters across all applications.
* If automatic hyperparameter search is used, the processing time shall be reported as part of the encoding time.

## Guidelines for providing models and data

1. train the network
   1. network input as will be read while training, ideally one file/folder, example script for reading the data
   2. labels such as applicable for different classes etc. used while training
   3. ideally, a dedicated training , tuning and test set, i.e. one large dataset for training the model, a smaller set for selecting hyperparameters or compression parameters, and an independent data set for testing
   4. the test data shall contain the annotations used for computing the performance metrics in accessible form, ideally output class labels etc. in a separate numpy array that is compared to the actual output of the network
2. original model and supporting code
   1. at least one trained model file is to be provided
   2. the Python script provided should support using an compressed model, i.e. with pruned or quantized weights. This can happen inside the script. i.e. keras.model.load(..)
   3. the Python script shall output the key performance metrics of running the model, ideally as numbers inside python (result.performance = 0.93)

# 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 the entire model when loaded in memory. R\_size shall be measured from the representation of the model used for inference.

## Compressed model size (Os\_size, Cs\_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 Os\_size. The size of the compressed model Cs\_size is measured as the sum of the size of components of the serialized model, using the same format for the original and compressed model where applicable. If the format used for the compressed model does not support the representation of specific data structures used in the proposal, those may be serialized in a custom format (in one or multiple files).

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

Here we compare the model before and after compression, denoted as O\_Per, R\_Per (performance of compressed model without fine-tuning/retraining) and RR\_Per (performance of compressed model without fine-tuning/retraining). In other words, O\_Per and R\_Per (RR\_Per) should be as close as possible. In order to ensure the same reference O\_Per independent of the deep learning framework used in an experiment, a single model is selected as a starting point for each task and also provided in a representation to allow for import in other frameworks.

One of R\_Per and RR\_Per must be reported, the second one is optional (if feasible, both values shall be reported). For methods which strongly rely on retraining, R\_per may not be meaningful, while RR\_Per is not available for methods not applying retraining. For methods applying compression and fine-tuning steps iteratively, R\_Per shall report the performance after the first compression, and RR\_Per after the last fine-tuning.

If feasible, the performance of the methods should be reported for at least three working points per application, covering a range of size vs. performance trade-offs that is as broad as possible.

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

## Runtime and memory complexity

Proponents are required to report the hardware and software configuration used for running the runtime measurements of the respective steps.

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

### Compression

Runtime and memory complexity of the compression step will be measured.

The following metrics shall be reported

* Runtime of compression of the model (as defined in Section 5), and runtime of retraining (if applicable, as defined in Section 5)
* Memory consumption of the model compression step

If the method uses fine-tuning/retraining after compression, the runtime of this part of the encoding process shall be reported separately.

### Decompression

Runtime and memory complexity of decompression step, i.e. loading and (if needed) decoding the model into the representation used for inference, will be measured.

The following metrics shall be reported

* Runtime of decompressing/loading the model (as defined in Section 5)
* Number of operations of decompressing the model (as defined in Section 5)
* Memory consumption of the model decompression step

### Inference

Runtime complexity of inference step using both the uncompressed and the reconstructed model will be measured.

The following metrics shall be reported

* Runtime of inference using the respective model(as defined in Section 5)
* Number of operations of inference using the model (as defined in Section 5)

## Incremental representation (C\_ratio)

NOTE: This section does not apply to the current experiments.

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 Applications

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/VGG16 https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| ResNet50 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/ResNet50 https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels.h5 |
| MobileNet v2 | Tensorflow / Keras | https://www.tensorflow.org/api\_docs/python/tf/keras/applications/MobileNet https://github.com/pytorch/vision/tree/master/torchvision/models |

### 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. For training and compression, only the training subset may be used. Results shall be reported for the validation subset, which may not be used for training, but for selection of hyperparameters.

The evaluation for image classification is expected to be conducted for a single center-crop with the following crop-sizes for network architectures. Center-cropping shall be done by resizing the image so that the shorter edge is 224+32 (interpolation=cv2.INTER\_CUBIC), and center cropping the resized image to (224x224).

|  |  |  |
| --- | --- | --- |
| VGG16 | ResNet50 | MobileNet |
| 224x224 | 224x224 | 224x224 |

The numbers are copied from the explanation in the reference implementation for Keras (except Alexnet) from the following link: <https://github.com/keras-team/keras-applications>

Alexnet crop is based on the implementation in the following link: <https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py>

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.

ONNX versions of the models are provided at <https://drive.google.com/file/d/1om3XeAv42sPdw1EcPh_3R7h6PbQAszwW/view?usp=sharing>

### Evaluation metrics

Top-5 classification performance shall be reported. Optionally, top-1 performance may be reported.

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

This task addressed visual object detection in images, using three different neural network models

### NN Models and models

1. Yolo-v3
   1. Pytorch model: <https://github.com/ultralytics/yolov3>
   2. pretrained weight based on MS COCO dataset COCO trainvalno5k dataset: <https://pjreddie.com/media/files/yolov3.weights>
   3. Dataset can be obtained by following bash file: <https://github.com/ultralytics/yolov3/blob/master/data/get_coco_dataset.sh>
2. Faster-RCNN
   1. Pytorch model: <https://github.com/ruotianluo/pytorch-faster-rcnn>
   2. Pretrained weight based on MS COCO trainval35k dataset using different backbone networks (ResNet50): <https://drive.google.com/drive/folders/0B7fNdx_jAqhtNE10TDZDbFRuU0E> (vgg16/finetunr\_from\_scratch/coco\_350k-490k.tar)
   3. Dataset can be obtained from the following link:

<https://github.com/rbgirshick/py-faster-rcnn/tree/master/models>

1. Mask-RCNN
   1. Keras/Tensorflow model: <https://github.com/matterport/Mask_RCNN>
   2. Pretrained weight based on MS COCO trainval35k dataset: <https://github.com/matterport/Mask_RCNN/releases/tag/v2.0>
   3. Dataset can be obtained from MS COCO requirement section of following link:

<https://github.com/matterport/Mask_RCNN>

### Evaluation metrics

Performance evaluation:

* Yolo\_v3 and Faster R-CNN: 0.5 IOU mAP
* Mask R-CNN: follow the python script for segmentation evaluation in cocoeval.py [https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py, 0.5](https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py,%200.5) IOU mAP 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. 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].

Please send a message to the reflector for information how to obtain the data set.

### 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 a filter in the VTM reference software codec.

### NN Model

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 .mat file for loading and testing.

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

### Test framework and data

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

Evaluation is mandatory for the all intra (AI) case, while testing for the random access (RA) case is optional. For the AI case, the output images to which the filters are applied will be provided to facilitate the evaluation. For this case, a standalone filter tool is provided.

The standalone filtering tool is provided at https://drive.google.com/open?id=13RbDKII0PTNFxIEdl5vxfTWauAAuY8Ns

The bitstreams of test sequences encoded with VTM-3.0 are available at https://drive.google.com/open?id=13RbDKII0PTNFxIEdl5vxfTWauAAuY8Ns

To decode the bitstream, VTM-3.0 can be built from the source code available at:

<https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM>

For example:

git clone <https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM> vtm-3.0 cd vtm-3.0 git checkout VTM-3.0

then in a build directory:

cmake /path/to/vtm-3.0 -G "target platform"

make

Run the decoding with the following command

Path\_to\_bin/DecoderApp -b str.bin -o dec.yuv > dec.txt

The subsection describes the details on a network model of CNN for replacing in-loop filter [3], using VTM 3.0 as a basis. 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.

|  |  |
| --- | --- |
| Saving/Definition of the weights in trained network (Tensorflow) | Loading of the trained network in Tensorflow |
| # weight definition  W1 = tf.Variable(tf.random\_normal([7, 7, 1, 64], stddev=0.01))  Layer1=tf.nn.conv2d(inputss, W1, strides=[1,1,1,1], padding='SAME')+b1 Layer1=tf.nn.relu(Layer1)  # save weight file with .mat file sess.run(W1) array1=W1.eval(sess) sio.savemat('W1', {'W1':np.float16(array1)}) | # Weight load (W1~W7, b1~b7) W11=sio.loadmat('W11.mat') W1=np.float32(W11['W1'])  W21=sio.loadmat('W21.mat') W2=np.float32(W21['W2']) |

### Evaluation metrics

The following metrics shall be applied:

* PSNR (required)
* BD-rate (only for RA case)

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 Keras are shown below. The model trained in Keras is represented as 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 Keras | Saving/loading of the trained network in Keras |
| def Encoder():  x = input\_img  with k.tf.device('/gpu:0'):  x = Conv2D(64, (3, 3), activation='relu', padding='same')(input\_img)  x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)  x = MaxPooling2D((2, 2), padding='same')(x)  x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)  x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)  encoded = MaxPooling2D((2, 2), padding='same')(x)  return Model(input\_img, encoded)  def Decoder():  dec\_input=Input(shape=(8,8,16))  with k.tf.device('/gpu:0'):  x = UpSampling2D((2, 2))(dec\_input)  x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)  x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)  x = UpSampling2D((2, 2))(x)  x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)  x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)  decoded = Conv2D(3, (3, 3), activation=None, padding='same')(x)  return Model(dec\_input, decoded)    encoder= Encoder()  decoder= Decoder() | autoencoder.save\_weights('imagecompression.hdf5')encoder.save\_weights('encoder\_weight.hdf5')decoder.save\_weights('decoder\_weight.hdf5')#load model&weightsload\_model('autoencoder.json')autoencoder.load\_weights('imagecompression.hdf5')encoder.load\_weights('encoder\_weight.hdf5') decoder.load\_weights('decoder\_weight.hdf5') |

The trained model is available at <https://drive.google.com/drive/folders/1d-MaaXebVxd8yGUCv44RWQvLAwzVm6JQ>

The part of the data to be used for parameter tuning is described at

<https://drive.google.com/drive/folders/1d-MaaXebVxd8yGUCv44RWQvLAwzVm6JQ>

### 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.

## UC16A Acoustic Scene Classification

The goal of acoustic scene classification is to classify a test recording into one of the provided predefined classes that characterizes the environment in which it was recorded — for example park, pedestrian street, metro station, seaside. We will use the DCASE2017 Task1 Data and the DCASE2018 baseline classification system.

### NN Models

DCASE 2019 Acoustic Scene Classification Baseline

* https://github.com/toni-heittola/dcase2019\_task1\_baseline
* *Acoustic features:* Analysis frame 40 ms (50% hop size) Log mel-band energies (40 bands)
* *Neural network:* Two convolutional layers with max pooling, two dense layers. Batch nom and dropout.

### Test framework and data

The TUT Acoustic Scenes 2017 dataset consists of recordings from various acoustic scenes, all having distinct recording locations. For each recording location, 3-5 minute long audio recording was captured. The original recordings were then split into segments with a length of 10 seconds. These audio segments are provided in individual files. There are a total of 15 acoustic scenes for the task. <http://dcase.community/challenge2017/task-acoustic-scene-classification>

### Evaluation metrics

For audio classification is classification accuracy.

### Evaluation Framework

The code as well as training, validation and test split is published here:

<https://drive.google.com/open?id=1jBWk_uyVjos8kQhnHY7k-reT-C9j9CoF>

After unzipping, run

python mpeg-task-uc16a.py eval -m mpeg\_model.h5

to use the model we trained on the development set. The -w parameter allows to use your own weights

mpeg-task-uc16a.py eval -m mpeg\_model.h5 -w my\_weights.h5

or

mpeg-task-uc16a.py test -m mpeg\_model.h5 -w my\_weights.h5

for generating the final number for the CE.

Eval and test are separate sets files, also separate from the training data.

In order to change the weights, use the hdf5 file provided or the keras methods:

from keras.models import load\_model

model = load\_model('mpeg\_model.h5')

wq = model.get\_weights()

wn = your\_method(wq)

model.set\_weights(wn)

model.save\_weights('my\_weights.h5')

### Evaluation Framework

The Training, Validation and Test split is published on google drive together with the code:

<https://drive.google.com/open?id=1jBWk_uyVjos8kQhnHY7k-reT-C9j9CoF>

After unzipping, run

python mpeg-task-uc16a.py eval -m mpeg\_model.h5

to use the model we trained on the development set. The -w parameter allows to use your own weights

mpeg-task-uc16a.py eval -m mpeg\_model.h5 -w my\_weights.h5

or

mpeg-task-uc16a.py test -m mpeg\_model.h5 -w my\_weights.h5

for generating the final number for the CE.

Eval and test are separate sets files, also separate from the training data.

In order to change the weights, use the hdf5 file provided or the keras methods:

from keras.models import load\_model

model = load\_model('mpeg\_model.h5')

wq = model.get\_weights()

wn = your\_method(wq)

model.set\_weights(wn)

model.save\_weights('my\_weights.h5')

## UC3 Natural language understanding (NLU) Q&A

As the topology of the networks used in NLU applications is very different from the others in the evaluation, it is of interest to check how these react to compression. IN particular, the “BERT” model, a multi-layer bidirectional Transformer encoder.

### Test Data

**S**tanford **Qu**estion **A**nswering **D**ataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or *span*, from the corresponding reading passage, or the question might be unanswerable.

<https://rajpurkar.github.io/SQuAD-explorer/>

### NN Models

**BERT**, or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. <https://arxiv.org/abs/1810.04805>.

<https://github.com/google-research/bert>

### Evaluation metrics

The typical evaluation metric is F1 measure computed from recall and precision. An evaluation script is provided with SQuAD.

### Evaluation baseline

A combined archive with an integrated test script using BERT and SQuAD is provided here:

<https://drive.google.com/drive/folders/17Gl7HSNpXu61qmRn-TKe70X45ItIVbmI>

# Runtime and Energy Consumption Measurement

Complexity of inference depends on the hardware the neural network model is being executed on. This makes the design of lossy compression schemes more difficult since, preferably, they ought to be platform agnostic. Moreover, benchmarking and comparing different compressed models becomes a tedious task, since they would need to be tested on a wide set of different hardware platforms. Hence, we define metrics that characterize the inference complexity in different levels of granularity, trading-off between simplicity/flexibility vs precision.

In this regards, we propose three levels of granularity

* total number of operations (required)
* approximate inference complexity per layer (optional)
* entire inference procedure (required)

## Total number of operations

In order to have a metric for the inference complexity which is flexible across a wide range of hardware platforms and neural network representations, we count the total number of atomic operations of the entire procedure. We understand atomic operations as those that are directly supported by the instruction set of the target platform (e.g., CPU, GPU) and are called from the lowest level of the implementation. The granularity of these atomic operations may very between platforms. This applies also to operations like conditions.

Proponents shall report the total number of atomics operations, aggregated across layers. Optionally, they may report the total number of operations of each layer independently.

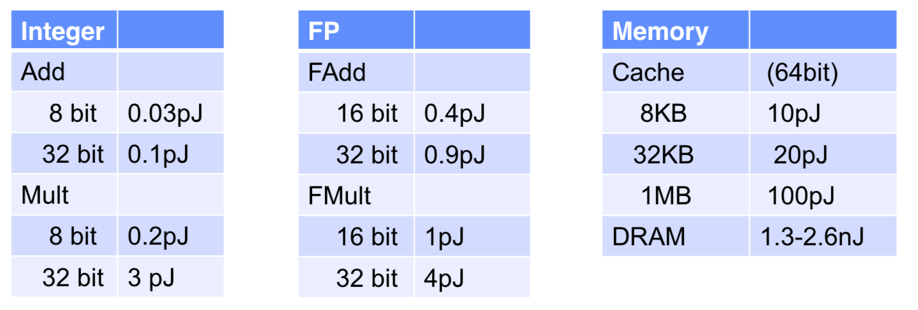
Optionally, data on how these operations are broken down into elementary operations shall be provided, for example:

* Addition
* Multiplication
* Read from memory, including size of the bitstream accessed
* Write into memory, including size of the bitstream accessed

For instance, a usual MAC operation would then be reported as: 1 add, 1 mul, 2 read and 1 write operation. By considering these 4 elementary operations independently we will be able to compare the inference complexity of different network representations in a more objective manner (e.g. consider comparison of the inference procedure between dense and sparse layers).

Weighting each type of operation by the respective runtime and energy cost allows for fast approximate comparison different models more specifically for a particular hardware. Proponents may optionally report runtime and/or energy costs for the respective operation on the target platform they are using.

For instance, rough energy numbers of different elementary operations for a 45nm CMOS [11] are



## Evaluate approximate inference complexity per layer

One of the goals of neural network compression is that the encoded network representation does not require decoding in order to run inference. Hence, newly proposed network representations may have different algorithms associated to them that calculate operations such as the dot product or convolutions. E.g., think of sparse representations and their respective dot product algorithms, such as the CSR format. Hence, directly evaluating the inference complexity by using common frameworks such as Keras or Caffe2 may be infeasible in a given time frame since it may require replacing the implementation of these operations with modified (and optimized) versions in these frameworks.

The inference complexity can be approximated by evaluating the runtime of calculating the preactivation values at each layer. Since a significant part of the computational complexity of inference is spent in calculating the preactivation values at each layer, capturing the complexity of these operations gives a good approximation of the whole inference procedure. This approximation can be done for each layer separately by running the code that calculates the preactivation values given the input activations and the weights. These measurements may use different implementations for the different layers, e.g., some may be implemented in a standard deep learning framework, while some may use a custom implementation of modified operations.

Hence, for the ease of implementation and fast benchmarking on different hardware platforms, the following procedure shall be applied:

For each layer, the input activations and weight bit-streams shall be reported, along with the associated binaries that calculate the preactivation values from them. The measurement shall be done by processing one sample at a time, and average over a sufficiently large number of samples from the respective test data set.

The measured runtimes (and optionally energy consumption) per layer shall be reported.

## Evaluate the entire inference procedure

In order to have and compare realistic measurements regarding the run-time/energy complexity of the inference procedure of different networks, the entire inference procedure needs to be implemented and benchmarked on the target hardware.

The measurement shall be done by processing one sample at a time, and average over a sufficiently large number of samples from the respective test data set.

The runtime and memory consumption of the respective step (compression, decompression, inference) shall be reported, and a specification of the properties of the target platform shall be provided. It is understood that the use of different implementations within frameworks/libraries that are optimized to a different degree will impact the overall runtime. Thus the measurements are complemented with the operations count.

# References

1. F. Bossen, et al, “JVET common test conditions and software reference configurations for SDR video” JVET document, JVET-L1010, Oct. 2018.
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>
11. M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 10–14.