|  |
| --- |
| **INTERNATIONAL ORGANIZATION FOR STANDARDIZATION ORGANISATION INTERNATIONALE DE NORMALISATION ISO/IEC JTC 1/SC 29/WG 5 MPEG JOINT VIDEO EXPERTS TEAM WITH ITU-T SG 16** |
| **ISO/IEC JTC 1 / SC 29 / WG 5 N 309** |
| **Sapporo, JP – 12–19 July 2024** |
| |  |  | | --- | --- | | **Source:** | **Convenor (Jens-Rainer Ohm)** | | **Title:** | **Description of algorithms version 8 and software version 10 in neural network-based video coding (NNVC)** | | **Type:** | **General** | | **Subtype:** | **Other** | | **Status:** | **Approved** | | **Date:** | **2024-10-11** | | **Expected Action:** | **Info** | | **Action due date:** | **N/A** | | **Pages:** | **46** (not including this cover page) | | **Email of convenor:** | **ohm @ ient . rwth-aachen . de** | | **Committee URL:** | **https://sd.iso.org/documents/ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-29/iso-iec-jtc-1-sc-29-wg-5** | |

|  |  |
| --- | --- |
| **Joint Video Experts Team (JVET)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29**  **35th Meeting: Sapporo, JP, 12–19 July 2024** | Document: JVET-AI2019 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Description of algorithms version 8 and software version 10 in neural network-based video coding (NNVC)** | | |
| *Status:* | Output document of JVET | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Franck Galpin  Yue Li  Dmytro Rusanovskyy  Jacob Ström  Liqiang Wang | Email: | [franck.galpin@interdigital.com](mailto:franck.galpin@interdigital.com)  [yue.li@bytedance.com](mailto:yue.li@bytedance.com)  dmytror@qti.qualcomm.com  [jacob.strom@ericsson.com](mailto:jacob.strom@ericsson.com) [liqiangwang@tencent.com](mailto:liqiangwang@tencent.com) |
| *Source:* | InterDigital, Bytedance, Qualcomm, Ericsson, Tencent. | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This document is version 8 of a description of algorithms and software for Neural Network-based Video Coding. Due to a lack of synchronization between the document and the software development, it corresponds with a software codebase released as NNVC 10. It includes the coding features and encoding methods implemented in NNVC software that are under coordinated exploration study by the Joint Video Exploration Team (JVET) of ITU-T VCEG and ISO/IEC MPEG as potential neural network video coding technology beyond the capabilities of VVC. The groups are working together on this exploration activity in a joint collaboration effort known as the Joint Video Exploration Team (JVET) to evaluate compression technology designs proposed by their experts in this area.

Contents

[Abstract 1](#_Toc169102202)

[1 Introduction 4](#_Toc169102203)

[2 Scope 4](#_Toc169102204)

[3 NNVC software usage 4](#_Toc169102205)

[4 Algorithm description of Neural Network-based Video Coding Software 6](#_Toc169102206)

[4.1 Low complexity operation point (LOP) neural network-based loop filter 6](#_Toc169102207)

[4.1.1 Neural network 6](#_Toc169102208)

[4.1.2 Residue scaling 8](#_Toc169102209)

[4.1.3 Combination with deblocking filters 8](#_Toc169102210)

[4.1.4 Inference details 8](#_Toc169102211)

[4.2 Very Low complexity operation point (VLOP) neural network-based loop filter 9](#_Toc169102212)

[4.2.1 Network Architecture and Training 9](#_Toc169102213)

[4.2.2 VLOP Inference 9](#_Toc169102214)

[4.3 High Operating Point (HOP) model 10](#_Toc169102215)

[4.3.1 Overview of HOP1 to HOP3 10](#_Toc169102216)

[4.3.2 HOP4 Network Architecture 10](#_Toc169102217)

[4.3.3 HOP4 Inference 12](#_Toc169102218)

[4.3.4 Model usage aspects 12](#_Toc169102219)

[4.3.5 Pre-processing of chroma 13](#_Toc169102220)

[4.3.6 Adaptive inference granularity 13](#_Toc169102221)

[4.3.7 Picture boundary padding 14](#_Toc169102222)

[4.3.8 Base QP adjustment 14](#_Toc169102223)

[4.3.9 Blending with DBF 15](#_Toc169102224)

[4.3.10 Encoder-only optimization 15](#_Toc169102225)

[4.3.11 Temporal filter 15](#_Toc169102226)

[4.3.12 Decoder complexity optimization 15](#_Toc169102227)

[4.3.13 Residual offset adjustment 16](#_Toc169102228)

[4.3.14 Inference details 16](#_Toc169102229)

[4.4 Neural network-based intra prediction 16](#_Toc169102230)

[4.4.1 Neural network inference 16](#_Toc169102231)

[4.4.2 Preprocessing and postprocessing 18](#_Toc169102232)

[4.4.3 Adaptation of the derivation of the list of MPMs 18](#_Toc169102233)

[4.4.4 Signaling of the neural network-based intra prediction mode 19](#_Toc169102234)

[4.4.5 Transformation of the context and the neural network prediction 20](#_Toc169102235)

[4.4.6 High complexity and low complexity versions of the neural network-based intra prediction mode 21](#_Toc169102236)

[4.5 Small ad-hoc deep learning (SADL) library 22](#_Toc169102237)

[4.6 Content-adaptive neural networks loop-filter 23](#_Toc169102238)

[4.6.1 Neural network 23](#_Toc169102239)

[4.6.2 Content adaptation 23](#_Toc169102240)

[4.6.3 Inference details 23](#_Toc169102241)

[4.7 Encoder 24](#_Toc169102242)

[4.8 Decoder 24](#_Toc169102243)

[4.9 Content-adaptive neural network post-filter 24](#_Toc169102244)

[4.9.1 Neural network 24](#_Toc169102245)

[4.9.2 Content adaptation 25](#_Toc169102246)

[4.9.3 Inference details 25](#_Toc169102247)

[4.10 Neural network-based super resolution 25](#_Toc169102248)

[4.10.1 Neural network for luma component 25](#_Toc169102249)

[4.10.2 GOP level encoding resolution decision 26](#_Toc169102250)

[4.10.3 Inference details 27](#_Toc169102251)

[5 Training description of Neural Network-based Video Coding Software 29](#_Toc169102252)

[5.1 Low operating point neural network-based loop filter 29](#_Toc169102253)

[5.2 Very Low operating point neural network-based loop filter set 29](#_Toc169102254)

[5.3 Neural network-based intra prediction 29](#_Toc169102255)

[5.4 Content-adaptive neural network post-filter 30](#_Toc169102256)

[5.5 Content-adaptive neural network loop-filter 30](#_Toc169102257)

[5.6 Neural network-based super resolution 30](#_Toc169102258)

[5.7 High Operating Point model training 31](#_Toc169102259)

[5.7.1 Overview 31](#_Toc169102260)

[5.7.2 Training characteristics 31](#_Toc169102261)

[5.7.3 Prerequisite 32](#_Toc169102262)

[5.7.4 Training Model stage I 33](#_Toc169102263)

[5.7.5 Training Model stage II 34](#_Toc169102264)

[5.7.6 Training stage III 35](#_Toc169102265)

[6 Legacy filters 36](#_Toc169102266)

[6.1 Neural network-based loop filter set 0 36](#_Toc169102267)

[6.1.1 Pre-processing and post-processing of chroma 36](#_Toc169102268)

[6.1.2 Neural network 36](#_Toc169102269)

[6.1.3 Combination with conventional filters 36](#_Toc169102270)

[6.1.4 Mode selection 37](#_Toc169102271)

[6.1.5 Base QP adjustment 37](#_Toc169102272)

[6.1.6 Encoder-only Optimization 38](#_Toc169102273)

[6.1.7 Inference details 38](#_Toc169102274)

[6.2 Neural network-based loop filter set 1 39](#_Toc169102275)

[6.2.1 Neural network for luma component 39](#_Toc169102276)

[6.2.2 Neural network for chroma component 39](#_Toc169102277)

[6.2.3 Temporal filter 40](#_Toc169102278)

[6.2.4 Adaptive inference granularity 40](#_Toc169102279)

[6.2.5 Parameter selection 40](#_Toc169102280)

[6.2.6 Residue scaling 41](#_Toc169102281)

[6.2.7 Combination with deblocking filter 41](#_Toc169102282)

[6.2.8 Encoder-only optimization 41](#_Toc169102283)

[6.2.9 Inference details 42](#_Toc169102284)

[6.3 Training Legacy Neural network-based loop filter set 0 42](#_Toc169102285)

[6.4 Training Neural network-based loop filter set 1 43](#_Toc169102286)

[6.4.1 Regular filters 43](#_Toc169102287)

[6.4.2 Temporal filter 44](#_Toc169102288)

[7 References 44](#_Toc169102289)

# Introduction

This document provides algorithm description, encoding methods, and training methods of the neural network-based coding tools implemented in Neural Network-based Video Coding (NNVC) software. The neural network-based tools (NN-based tools) are to enhance or replace conventional modules in the existing VVC design [2]. The implementation of NN-based tools in NNVC are based on Small Ad-hoc Deep Learning (SADL) library [3]. It is recommended to refer to the software repository of NNVC [4] for the detailed information of VTM-11.0, which is the base of NNVC and the SADL documentation for the detailed usage of SADL.

# Scope

The NNVC reference software is provided to demonstrate a reference implementation of encoding techniques and the decoding process, as well as the training methods for neural network-based video coding explored in JVET. The reference software can be accessed via

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

The branch of the software is VTM-11.0\_nnvc (the master branch being the original VTM).

This document provides an algorithm description, an encoder-side description, and a training description of the NNVC, which serves as a tutorial for the algorithm and encoding model implemented in the NNVC software, as well as the training method of the tools in NNVC software. The purpose of this document is to share a common understanding of the coding features and the reference encoding methods supported in the NNVC software, in order to facilitate the assessment of the technical impact of new technologies during the exploration work.

# NNVC software usage

We describe the several variants which can be used in NNVC. Combination of these models can be used, wherever it makes sense. Current official models for anchors are in bold.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variant** | **Command-line** | **Model name** | **Comment** | **Reference** |
| **NN intra** | --NnIntraPred=1 | --PrefixAbsolutePathsToGraphsOutput= models/intra | Used for NNVC anchor | JVET-AI0130 |
| LOP2 | --NnlfOption=1 | --NnlfModelName= nnlf\_lop2\_model\_int16.sadl | Anchor ILF method for JVET-AF, AG, AH meetings | JVET-AF0043 |
| LOP2 b | --NnlfOption=1 | --NnlfModelName= nnlf\_lop2\_AH0042 \_model\_int16.sadl | Improved training for LOP2 | JVET-AH0042 |
| **LOP3** | --NnlfOption=3 | --NnlfModelName= nnlf\_lop3\_model\_int16.sadl | NNVC anchor since JVET-AI | JVET-AH0080 |
| LOP3 AH0081 | --NnlfOption=3 | --NnlfModelName= nnlf\_lop3\_AH0081\_model\_int16.sadl | LOP3 with 3 stages training | JVET-AH0081 |
| VLOP | --NnlfOption=1 | --NnlfModelName= nnlf\_vlop\_model\_int16.sadl |  | JVET-AH0051 |
| VLOP2 | --NnlfOption=3 | --NnlfModelName= nnlf\_vlop2\_model\_int16.sadl | VLOP2 architecture aligned to LOP3 | JVET-AI0107 |
| HOP1 | --NnlfOption=1 | --NnlfModelName= nnlf\_hop\_model\_int16.sadl |  | JVET-AF0041 |
| HOP3 | --NnlfOption=1 | --NnlfModelName= nnlf\_hop3\_model\_int16.sadl |  |  |
| HOP4 | --NnlfOption=4 | --NnlfModelName= nnlf\_hop4\_model\_int16.sadl |  | JVET-AH0205 &  JVET-AH0189 |
| HOP5 | --NnlfOption=4 | --NnlfModelName= nnlf\_hop5\_model\_int16.sadl |  | JVET-AI0172 |
| LOP2 adaptive | --NnlfOption=1 | Encoder and decoder:  --NnlfModelName= nnlf\_lop2\_model\_int16.sadl  Encoder only (split RA):  --NnfuEnabled=1  --NumNnfus=1  --NnfuPayloadFileName0= /path/to/nnr/bitstream  --NnfuModelFileName0= /path/to/sequence/qp/segment/model  Decoder only:  --NnfuOutputFileStem= </path/to/file/stem> --NnlfArch=lop2 |  | JVET-AH0096 |
| LOP3 adaptive | --NnlfOption=3 | Encoder and decoder:  --NnlfModelName= nnlf\_lop3\_model\_int16.sadl  Encoder only (split RA):  --NnfuEnabled=1  --NumNnfus=1  --NnfuPayloadFileName0= /path/to/nnr/bitstream  --NnfuModelFileName0= /path/to/sequence/qp/segment/model  Decoder only:  --NnfuOutputFileStem= </path/to/file/stem> --NnlfArch=vlop |  | JVET-AI0111 |
| VLOP adaptive | --NnlfOption=1 | Encoder and decoder:  --NnlfModelName= nnlf\_vlop\_model\_int16.sadl  Encoder only (split RA):  --NnfuEnabled=1  --NumNnfus=1  --NnfuPayloadFileName0= /path/to/nnr/bitstream  --NnfuModelFileName0= /path/to/sequence/qp/segment/model  Decoder only:  --NnfuOutputFileStem= </path/to/file/stem> --NnlfArch=vlop |  | JVET-AI0157 |
| NNSR | -c nn-based/nnsr/nnsr\_classAx\_sx.cfg | --NnsrModelName=super\_resolution/NNVC\_SR\_int16.sadl | Different options for A1 and A2 |  |
| RPR | -c nn-based/nnsr/nnsr\_classAx\_sx.cfg | --NnsrModelName=default\_rpr | Different options for A1 and A2 |  |
| NNPF | --SEINNPFCEnabled=1  --SEINNPFAEnabled=1 | --LCModelPath= models/NnlfSetLC/LC\_int16\_model0.sadl, models/NnlfSetLC/LC\_int16\_model1.sadl, models/NnlfSetLC/LC\_int16\_model2.sadl, models/NnlfSetLC/LC\_int16\_model3.sadl  --NnpfModelPath=/path/to/sequence/qp/segment/model | Check cfg/nn-based/nnpf/ for full params | JVET-AC0055 |

Some deprecated (not maintained) models are also available:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variant | Command-line | Model name | Comment | Reference |
| LOP1 | --NnlfOption=12 |  |  |  |
| Filterset0 | --NnlfOption=10 | --NnlfModelName=NnlfSet0\_model\_int16.sadl |  |  |
| Filterset1 | --NnlfOption=11 | --NnlfModelName= |  |  |
|  |  |  |  |  |

# Algorithm description of Neural Network-based Video Coding Software

## Low complexity operation point (LOP) neural network-based loop filter

### Neural network

The network structure of the low complexity operation point CNN based loop filter is shown in Figure 1. The inputs to the loop filter are reconstructed luma and chroma samples (Rec), predicted luma and chroma samples (Pred), boundary strength information for luma and chroma (BS), base QP (QPbase), slice QP (QPslice), and block prediction information (IPB).

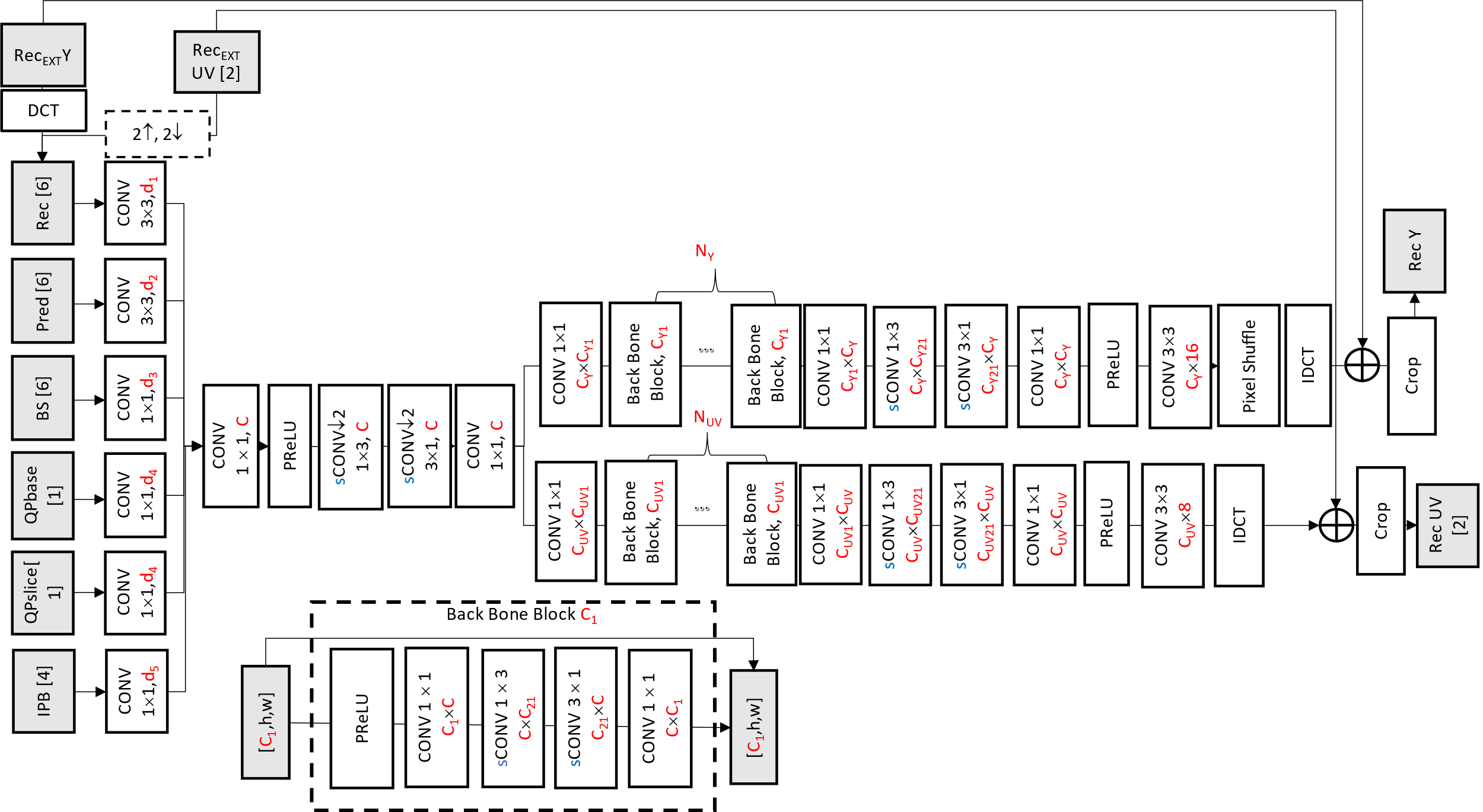


Figure . Architecture of low complexity CNN filter set.

The luma inputs are transformed, i.e., an input luma block of size WxH is transformed by a 2x2 DCT-II transform for each 2x2 subblock and reshaped into (W/2)x(H/2)x4, where 4 represents four frequency channels. The transformed luma samples are then concatenated with the chroma U and V channels to form a size of (W/2)x(H/2)x6. DCT transformation and planes expansion are applied to luma pixels of Rec, Pred, BS and IPB inputs, these are shown in Fig. 1 with red border blocks.

An example of reconstruction samples with size 144x144 is shown in Figure 2. The input luma of size 144x144x1 is transformed and reshaped into 72x72x4. For chroma reconstruction samples, because of upsampling of 2, each 2x2 subblock is constant and there is no need to apply transform. Thus, it can be directly downsampled by 2 to get to 72x72x2. The upsampling of 2 in chroma is kept to retain compatibility with the current training, due to that in the training data, the chroma data is already upsampled and saved together with luma. In a more efficient implementation, both upsampling and downsampling can be skipped with identical results.

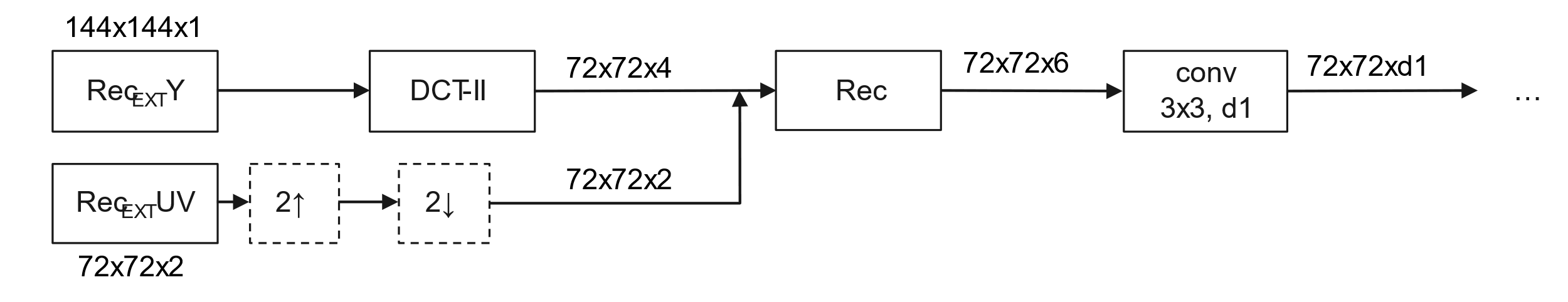


Figure . An example of applying the transform and reshaping to the reconstruction samples.

For QPbase and QPslice, since they are constant, there is no need to apply transform and they are directly reshaped to (W/2)x(H/2)x1. For IPB, as there is only luma information, it is transformed and reshaped into (W/2)x(H/2)x4.

The transformed inputs are processed by convolution layers with kernel sizes of 3x3 or 1x1 and concatenated for fusion and transition. In the fusion and transition module, separable convolutions of 1x3 and 3x1 and downsampling with a factor of 2 are used. The network then splits into two branches, one branch for luma and one branch for chroma. Each branch consists of sequential backbone blocks, and each backbone block consists of a PReLU, a convolution layer with a 1x1 kernel, separable convolution layers with 1x3 and 3x1 kernels, and a convolution layer with a 1x1 kernel. At the output side, PixelShuffle is applied to luma output, and inverse DCT is applied to both luma and chroma, such that the outputs get back to the same spatial size as the inputs.

### Residue scaling

When a NN filter is being applied to reconstructed pictures, a scaling factor is derived and signaled for each color component in the slice header. The derivation is based on least square method. The difference between the input samples and the NN filtered samples (residues) are scaled by the scaling factors before being added to input samples.

### Combination with deblocking filters

As shown in Figure 3, the reconstructed samples before Deblocking Filter are fed into the low complexity NN filter (NNLF), then final filtered samples are generated by blending the result of NNLF and Deblocking Filter.

=

A white circle with black text

Description automatically generated with low confidence

Figure . Parallel fusion of the NNLF and Deblocking Filter’s outputs

### Inference details

SADL (see Section 3.5) is used for performing the inference of the CNN filters. Both floating point-based and fixed point-based implementations are supported. In the fixed-point implementation, both weights and feature maps are represented with int16 precision using a dynamic quantization method. The network information in the inference stage is provided in Table 1.

Table 1 Network Information of low complexity filter set in Inference Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Total Parameter Number | 205108 |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 0.4 MB  int: 0.2 MB |
| Multiply Accumulate (kMAC/pixel) | 16.9 |
| Optional |  |  |
| Total Conv. Layers | 96 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 128x128, 256x256 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

## Very Low complexity operation point (VLOP) neural network-based loop filter

NN-based ILF of a very low complexity (5kMAC/pixel) was proposed adopted into NNVC. The network architecture is aligned to LOP3, and the inference uses the same implementation as LOP3.

### Network Architecture and Training

The VLOP2 NN architecture and its parameters are shown in Figure 4. VLOP2 is defined as a modification of the LOP3 architecture by adjusting the number of channels and backbone blocks to achieve a target complexity level. Similarly to LOP3, DCT and panels expansion is applied to luma pixels of Rec, Pred, BS and IPB inputs.

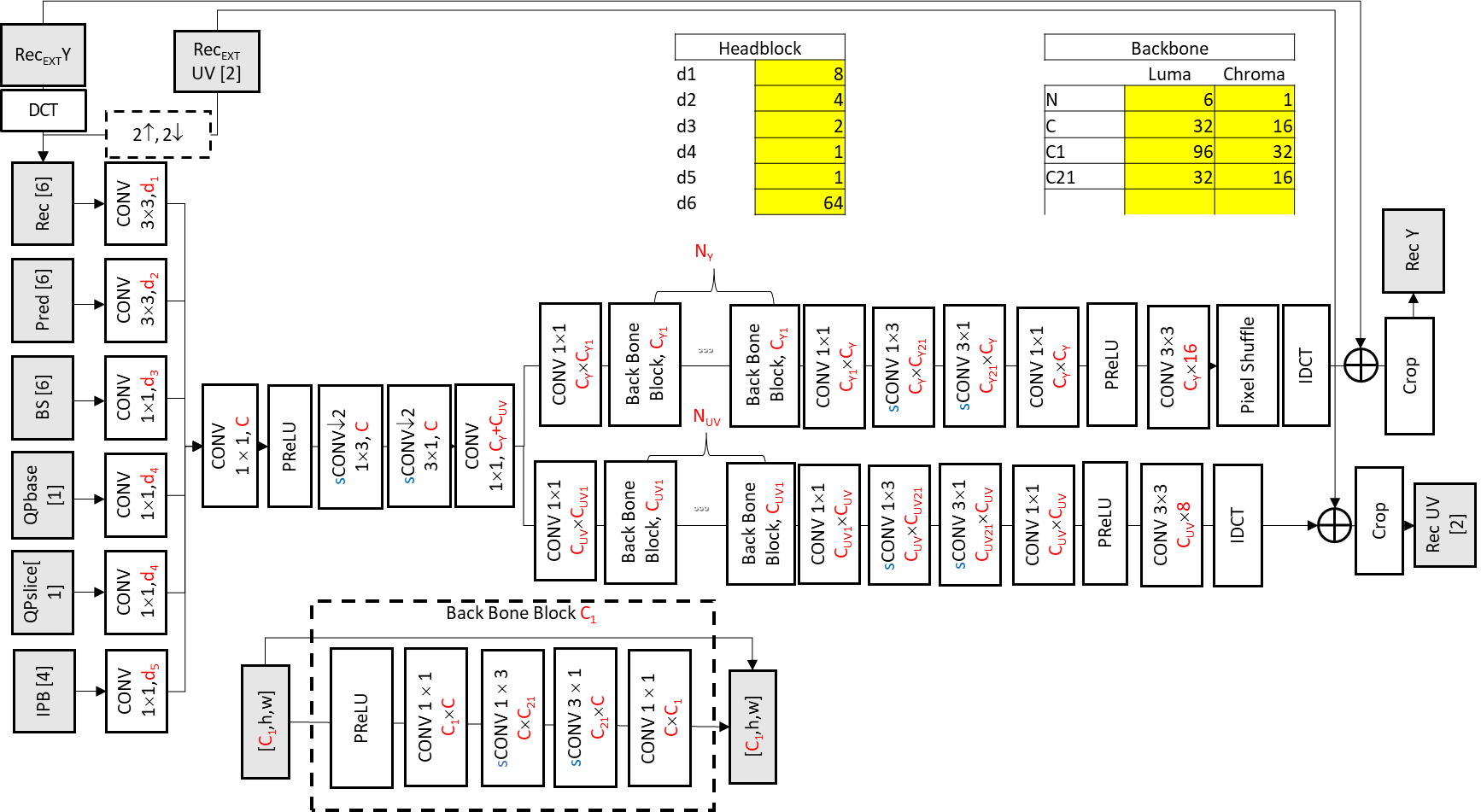


Figure . Architecture of NN based ILF for Very Low Operation Point (5kMAC/pixel).

VLOP2 model available in the reference NNVC SW has been trained using Stage 3 LOP2 dataset following an improved LOP2 training procedure, see documents JVET-AI0107 for details.

### VLOP Inference

VLOP2 inference process is aligned with the LOP3 inference. Both floating point and fixed-point arithmetic are supported, with respective models (nnlf\_vlop2\_model\_float.sadl, nnlf\_vlop2\_model\_int16.sadl) available in the reference NNVC SW. The VLOP2 network information in the inference stage is provided in Table 2.

Table 2 Network Information of low complexity filter set in Inference Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Total Parameter Number | 60686 |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 0.24 MB  int: 0.12 MB |
| Multiply Accumulate (kMAC/pixel) | 5.16 |
| Optional |  |  |
| Total Conv. Layers | 50 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 128x128, 256x256 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

## High Operating Point (HOP) model

### Overview of HOP1 to HOP3

More details of HOP design, training and results can be found in:

* **HOP design and training choices:** [JVET-AD0380](https://jvet-experts.org/doc_end_user/current_document.php?id=12944) [5] “BoG report on NN-filter design unification”.
* **Training progress:** [JVET-AE0042](https://jvet-experts.org/doc_end_user/current_document.php?id=12990) [6] “AhG14 & AHG11: Report on AhG teleconference on high operation point (HOP) unified filter training”.
* **HOP training results:** [JVET-AE0191](https://jvet-experts.org/doc_end_user/current_document.php?id=13154) [7] “AhG11: EE1-0 High Operation Point model”.
* **HOP training procedure and results:** [JVET-AE0289](https://jvet-experts.org/doc_end_user/current_document.php?id=13252) [8] “AhG11: HOP training process and models”.
* **HOP official models for partial training 2:** [JVET-AE0291](https://jvet-experts.org/doc_end_user/current_document.php?id=13254) [9] “AhG11: Performance of the NNVC HOP with quantized Stage II model”.
* **HOP full results:** [JVET-AF0041](https://jvet-experts.org/doc_end_user/current_document.php?id=13285) [10] AhG11: HOP full results.
* **HOP2 architecture:** Retraining of HOP using [JVET-AF0155](about:blank), [JVET-AF0180](about:blank).
* **HOP3 architecture:** [JVET-AG0174](https://jvet-experts.org/doc_end_user/current_document.php?id=13730).

### HOP4/HOP5 Network Architecture

NN-based ILF of a High Operational Point (HOP), targeting complexity of 470kMAC/pixel, was initially developed as Unified Filter Archiecture in JVET-AD0380 and adopted to NNVC (HOP1) from JVET-AF041. HOP4 is an evolution development of the Unified Filter Architecture and reuses its training and inference implementations.

In this architecture, input data consisting of reconstructed YUV samples (with UV data being spatially upsampled to the Y resolution) and supplementary data: prediction information, deblocking parameters (BS), quantization (QPBase, QPSlice) and coding mode information, are being processed in parallel by the HeadBlock (HB) and concatenated. The output of the HB is being processed by Fusion and Transition Block (FTB) that implements data fusion and spatial downsampling. Output of the FTB undergoes a processing by the Backbone consisting of a sequence of Backbone Blocks (BB). Filtering is finalized by Tailblock and luma/chroma separation.

HOP4 is defined as a modification of the HOP3 architecture with introduction of the Attention mechanism, implementation of which is shown in Figure 6. Its backbone consists of 24 backbone blocks (BB) with two of these blocks (id=8 and id=15) being enhanced with Attention (BB+A). The modules affected by Attention mechanism introduction are shown in red color in Figure 5.

HOP5 is a complexity/performance optimization of the HOP4 architecture. Comparing to the HOP4, 3x3 group convolution in the attention block is set to a group of 8, and the number of residue blocks reduced to N=23 and resulting complexity of 466 kMAC/ pixel. No changes to the inference process, comparing to HOP4.

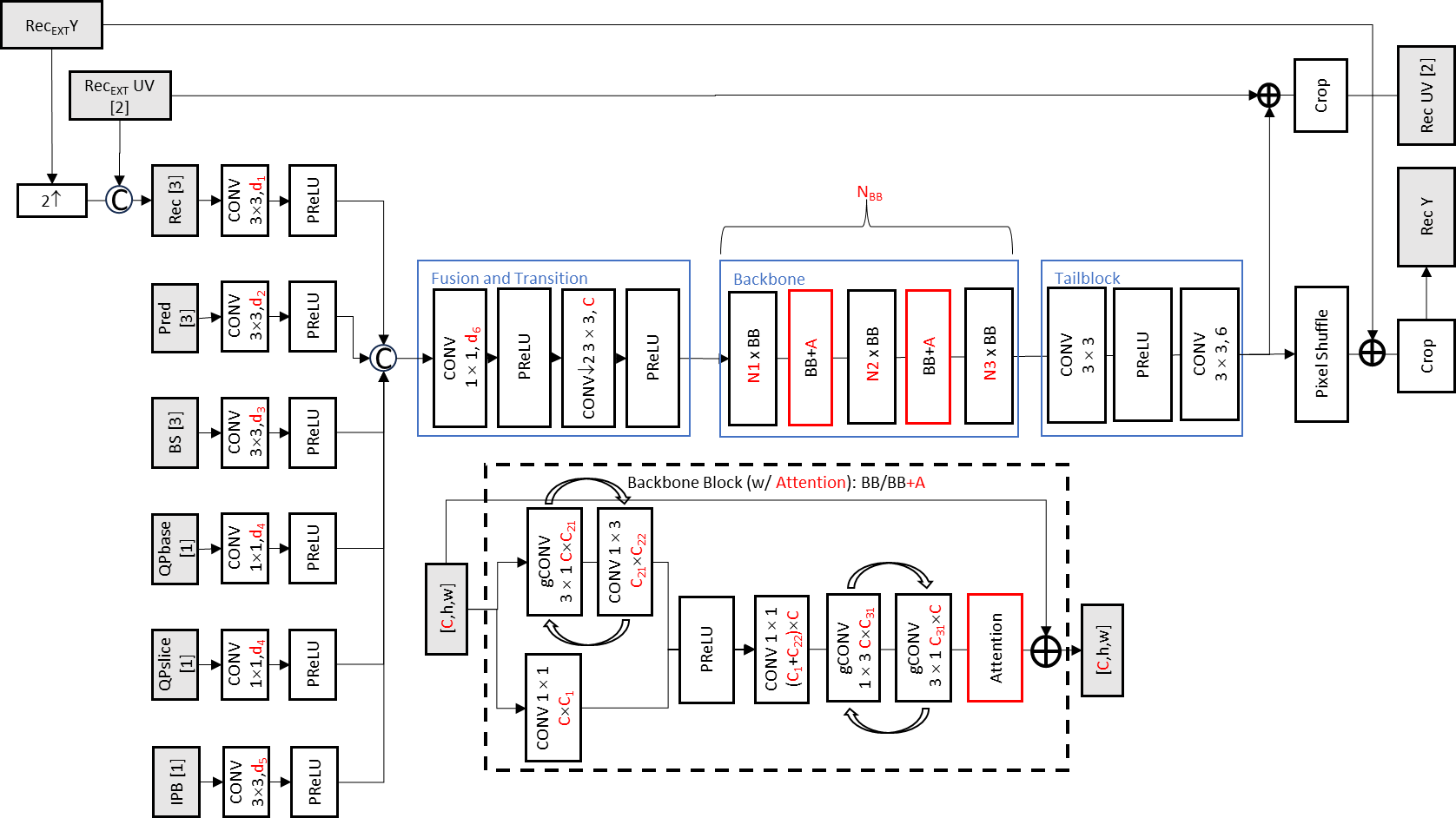


Figure . Architecture of NN based ILF for Very Low Operation Point (5kMAC/pixel).

A diagram of a machine

Description automatically generated

Figure . Design of the Attention block.

HOP4 model available in the reference NNVC SW has been trained using Stage 3 HOP dataset.

Table 3 **NN Filter network structure aspects**

|  |  |
| --- | --- |
|  | Unified  high tier filter |
| Joint YUV | √ |
| Intra = Inter model | √ |
| Prediction | √ |
| BS | √ |
| QP base | √ |
| QP slice | √ |
| IPB | √ |
| CONV for side Info | √ |
| Variable number Channels | √ |
| Long Skip connection | √ |
| Long/wide Activation | √ |
| Number ResBlocks | 24 |
| Conv Decomposed | √ |
| multi-scale feature extraction | √ |
| Number of channels | ×16 |
| Training Environment | pytorch 1.9 |

### HOP4 Inference

HOP4 inference is based on Unified Filter Architecture and using SADL interface. Both floating point and fixed-point arithmetic are supported, with respective models (nnlf\_hop4\_model\_float.sadl, nnlf\_hop4\_model\_int16.sadl) are being available in the reference NNVC SW. The HOP network information in the inference stage is provided in Table 4.

Table 4 Network Information of low complexity filter set in Inference Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Total Parameter Number | 2.9M |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 5.8 MB  int: 2.9 MB |
| Multiply Accumulate (kMAC/pixel) | 476 |
| Optional |  |  |
| Total Conv. Layers | 198 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 128x128 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

### Model usage aspects

The Table 5 gives the model application characteristics.

Table 5 **NN Filter interface aspects**

|  |  |
| --- | --- |
|  | Unified  Filter |
| Pre-processing and post-processing of chroma | √ |
| Enable DBF/SAO | Both |
| Blending with the DBF/SAO | DBF(\*) |
| Slice-level/Block-level blending with adaptive scale factor | √ |
| Slice-level/Block-level blending with fixed scale factor | √ |
| Separate adaptive scale factor for U and V | √ |
| Base QP adjustment | (-5, +5) |
| Encoder-only Optimization | × |
| Temporal filter | × |
| Adaptive inference granularity (128 or 256 block implicit) | √ |
| Configurable inference size (64 or 128 block explicit) | √ |
| configurable block extension size | √ |
| configuration QP params number | √ |
| block-level QP adaptation | √ |

(\*) NNLF comes after De-block but before SAO

### Pre-processing of chroma

In the unified HOP filter, the filter with a single model is designed to process three components. Since the resolutions of luma and chroma are different, pre-processing steps are introduced to up-sample chroma components as shown in Figure. 1. In the resampling process, the nearest-neighbor interpolation method is used.

A diagram of a diagram

Description automatically generated with medium confidence

*Figure. 4 the pre-processing unit*

### Adaptive inference granularity

Unified Filter Architecture defined in JVET-AD0380 introduced implicit, content-adaptive block size selection for the inference process, called inference granularity. Adaptive inference granularity with block size selection {128x128 or 256x256} as function of QP, Slice Type and picture width implemented through identical derivation process at the encoder and decoder sides. The larger inference block size is utilized for the Intra-coded Slice, as well as for Inter-coded Slice with picture width being larger than 832 pixels, and QP being above 30. The basic inference size (set as 128 by default) and block extension (set as 8 by default) in the inference region could be be specified at encoder side.

The HOP4 filter does not utilize implicit inference granularity and perform inference with a fixed block sizes.

### Picture boundary padding

To mitigate the boundary artifacts and reduce distortion, inference block is extended to include more samples from neighboring blocks. However, for inference blocks located at the picture boundary, neighboring blocks may not exist. In this case, the extended samples of inference block are padded with zero value.

### Base QP adjustment

Each slice or block could determine whether to apply the CNN-based filter or not. When the CNN-based filter is determined to be applied to a slice/block, which conditional parameter from a candidate list including two candidates derived from QP could be further decided. Denote the sequence level QP as q, the candidate list includes conditional parameters {Param\_1, Param\_2}. For low temporal layers, Param\_1 = q, Param\_2 = q5. For high temporal layers, Param\_1 = q, Param\_2 = q5. In other words, the second candidate is different across different temporal layers.

The selection process is based on the rate-distortion cost at the encoder side. Indication of on/off control as well as the conditional parameter index, if needed, are signalled in the bitstream. Figure. 6 shows the parameter selection of unified filter at encoder and decoder sides. All blocks in the current frame need to be processed with all conditional parameters first. Then all costs, i.e. Cost\_0, Cost\_1, ..., Cost\_N+1, are calculated and compared against each other to achieve optimum rate-distortion performance. In Cost\_0, CNN-based filter is prohibited for all blocks. In Cost\_i, {i = 1, 2, ..., N}, the parameter Param\_i is used for all blocks. In Cost\_N+1, different blocks may prefer different parameters, and the information regarding whether to use CNN-based filter or which parameter to be used is signaled for each block. At decoder side, whether to use CNN-based filter or which parameter to be used for a block is based on the Param\_Id parsed from the bit-stream as shown in Figure. 6 (b).

Note that for all-intra configuration, parameter selection is disabled while filter on/off control is still preserved. A shared conditional parameter is used for the two chroma components to ease the burden in worst case at decoder side. In addition, the max number of conditional parameter candidates, i.e. N, could be specified at encoder side (N = 2 by default).



*Figure. 6*. *(a)* *Parameter selection at encoder side. (b) Parameter selection at decoder side*.

In current implementation, the following parameters are used:

* param\_1: do not use the CNN
* param\_2: use CNN with qpOffset = 0;
* param\_3: use one CNN with either qpOffset = -5 or +5
* param\_4: us block-level CNN with qpOffset = 0/-5/+5

### Blending with DBF

Samples filtered by the deblocking filter and the NN filter are blended together via the following equation, where and refer to the outputs of NN filtering and deblocking filtering respectively, while stands for the blending weight.

=

There are four candidates, i.e. 1, 0.75, 0.5 and adaptive weight, for the blending weight. The adaptive weight is derived using least square method and signaled for each color component in the slice header.

### Encoder-only optimization

For a better estimation of rate-distortion (RD) cost in the case the NN filter is used, the proposed encoder introduces NN-based filtering into the rate-distortion optimization (RDO) process of partitioning mode selection. Specifically, a refined distortion is calculated by comparing the NN filtered samples and the original samples. The partitioning mode with the smallest rate-refined distortion cost is selected as the optimal one. To reduce complexity, several fast algorithms are applied. First, NN model is simplified by using a smaller number of residual blocks. Second, parameter selection is not allowed for the NN filtering in the RDO process Third, the proposed technique is only applied to the coding units with height and width no larger than 64. The NN filter used in the RDO process is also implemented with SADL using fixed point-based calculation. This NN-based encoder-only method is disabled by default.

### Temporal filter

Pictures could be processed by an additional NN-based in-loop filter, namely temporal fitter, which takes collocated blocks from the first picture in both reference picture lists to improve performance. The two collocated blocks are directly concatenated and fed into the network. When enabling temporal filtering feature, the temporal filter is applied to the luma component of pictures in three highest temporal layers, while the regular luma and chroma filters are used for other cases. By default, this temporal filtering feature is disabled.

### Decoder complexity optimization

To optimize the decoder complexity of unified filter, an encoder-only coding tool is proposed to adaptively skip blocks with small filter gains. Specifically, the filter gains of different blocks processed by NNLF are measured using Euclidean distance and calculated by

Where denotes the filter result of NNLF, denotes the unfiltered reconstruction and denotes the original image without compression. denotes the filter gain of current block and means some filter gains can be achieved by using NNLF. The larger the absolute value of , the greater the filter gain. This NN-based encoder-only tool is disabled by default.

The algorithm process is shown as follows.

* Check the block-level NNLF on/off is enabled.
* Calculate of each block.
* Sort all the blocks with from small to large filter gains.
* Calculate the gain ratio of each block: , where .
* Compare all with . If , disable NNLF for current block. Specifically, is a predefined threshold.

### Residual offset adjustment

When a NN filter is being applied to reconstructed pictures, a residual offset value is selected and signaled for each color component in the slice header. The offset value candidates are {1, 2}. The residual of NNLF’s output is adjusted by reducing the magnitude of the residual at each pixel by this small offset value before being added to input samples.

### Inference details

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| Framework: | SADL |
| Number of Parameters (Each Model) | 1 model: 1.45M |
| Parameter Precision (Bits) | int: 16 |
| Memory Parameter (MB) | 2.8MB |
| Multiplay Accumulate (MAC)/pixel | HOP1:477 kMAC/pix (assuming block-level input)  HOP3:466 kMAC/pix (assuming block-level input)  HOP4: see previous table |
| Optional |  |  |
| Total Conv. Layers | 154 |
| Total FC Layers | 0 |
| Batch size: | 1 |
| Patch size | HOP1 to HOP3: 128128, 256256 (+8 border) |

## Neural network-based intra prediction

### Neural network inference

The neural network-based intra prediction mode contains neural networks, each predicting blocks of a different size in . The neural network predicting blocks of size is denoted where gathers its parameters. For a given block , takes a preprocessed version of the context made of rows of reference samples located above this block and columns of reference samples on its left side to provide . The application of a postprocessing to yields a prediction of , see Figure 3. Besides, returns two indices and . denotes the index characterizing the LFNST kernel index and whether the primary transform coefficients resulting from the application of the DCT-2 horizontally and the DCT-2 vertically to the residue of the neural network prediction are transposed when , , see Figure 3. Furthermore, gives the index of the VVC intra prediction mode (PLANAR or DC or directional intra prediction mode) whose prediction of from the reference samples surrounding best represents , see Figure 3.

Diagram

Description automatically generated

Figure 5: prediction of the current block from the context of reference samples around via the neural network-based intra prediction mode. Here, and

If :

otherwise:

if

otherwise:

if

otherwise:

If , . Otherwise,

If , . Otherwise, .

### Preprocessing and postprocessing

#### Preprocessing of the context of the current block

The “preprocessing” shown in Figure 3 consists in the four following steps.

* The mean of the available reference samples in , see Figure 4,is subtracted from.
* If the neural network predicting the current block is in floats, the reference samples in the context are multiplied by, being the internal bitdepth, i.e. in VVC. Otherwise, the reference samples in the context are multiplied by , denoting the input quantizer.
* All the unavailable reference samples in , see Figure 4, are set to .
* The context resulting from the previous step is flattened, yielding , a vector of size .

Chart, diagram, box and whisker chart

Description automatically generated

Figure 6: decomposition of the context of reference samples surrounding the current block into the available reference samples and the unavailable reference samples . Here, and . In the illustrated case, the number of unavailable reference samples reaches its maximum value.

#### Postprocessing of the neural network prediction

The “postprocessing” depicted in Figure 3 consists in reshaping the vector of size into a rectangle of height and width , dividing the result of the reshape by , adding the mean of the available reference samples in the context of the current block, and clipping to Therefore, the postprocessing can be summarized as

### Adaptation of the derivation of the list of MPMs

When creating the MPM list of a given luma CB, if the “left” luma CB is predicted via the neural network-based intra prediction mode, the neural network-based mode index can be replaced by the returned during the prediction of the “left” luma CB and become a candidate index to be put into the MPM list. Similarly, if “above” luma CB is predicted via the neural network-based intra prediction mode, the neural network-based mode index can be replaced by the returned during the prediction of the “above” luma CB and become a candidate index to be inserted into the MPM list.

### Signaling of the neural network-based intra prediction mode

#### Signaling of the neural network-based intra prediction mode in luma

For the current luma CB whose top-left pixel is at position in the current luma channel, the intra prediction mode signaling in luma is split into two cases.

* If , *nnFlag* appears in the intra prediction mode signaling in luma. *nnFlag* means that the neural network-based intra prediction mode is selected to predict the current luma CB and END. *nnFlag*  means that the neural network-based intra prediction mode is not selected to predict the current luma CB, then the regular intra prediction mode signaling in luma, denoted , applies, see Figure 5.
* Otherwise, the regular intra prediction mode signaling in luma applies.

Note that, in the case “ if the context of the current luma CB goes out of the bounds of the current luma channel, i.e. , the neural network-based intra prediction is replaced by PLANAR.

.

Diagram

Description automatically generated

Figure 7: intra prediction mode signaling for the current luma CB framed in orange in dashed line. The coordinates of the pixel at the top-left of this CB are The bin value of a nnFlag value appears in bold gray. Here, , , , and .

#### Signaling of the neural network-based intra prediction mode in chroma

For the current pair of chroma CBs having top-left pixel at position in the current pair of chroma channels, the intra prediction mode signaling in chroma is split into two cases.

* If the luma CB collocated with this pair of chroma CBs is predicted by the neural network-based intra prediction mode:
  + If the DM becomes the neural network-based intra prediction mode
  + Otherwise, the DM is set to PLANAR.
* Otherwise:
  + If , *nnFlagChroma* appears in the intra prediction mode signaling in chroma. *nnFlagChroma* is placed before the DM flag in the decision tree of the intra prediction mode signaling in chroma. *nnFlagChroma* means that the neural network-based intra prediction mode is selected to predict the current pair of chroma CBs and END. *nnFlagChroma*  means that the neural network-based intra prediction mode is not selected to predict the current pair of chroma CBs, then the regular intra prediction mode signaling in chroma resumes from the DM flag.
  + Otherwise, the regular intra prediction mode signaling in chroma applies.

Note that, in the case where “and the case where “”, if the context of the current chroma CB goes out of the bounds of the current chroma channel, i.e. , the neural network-based intra prediction is replaced by PLANAR.

### Transformation of the context and the neural network prediction

For a given block, if , it is possible that the neural network-based intra prediction mode must predict this block but the neural network-based intra prediction mode does not contain . In this case, the context of the current block can be down-sampled vertically by a factor and/or down-sampled horizontally by a factor and/or transposed before the step called “preprocessing” in Figure 3. Then, the prediction of the current block can be transposed and/or up-sampled vertically by the factor and/or up-sampled horizontally by the factor after the step called “postprocessing” in Figure 3. The transposition of the context of the current block and the prediction, , and are chosen so that a neural network belonging to the neural network-based intra prediction mode is used for prediction, see Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| height and width of the block to be predicted |  |  | transposition | neural network used for prediction |
|  |  |  | no |  |
|  |  |  | no |  |
|  |  |  | yes |  |
|  |  |  | no |  |
|  |  |  | yes |  |
|  |  |  | no |  |
|  |  |  | yes |  |
|  |  |  | no |  |
|  |  |  | no |  |
|  |  |  | yes |  |
|  |  |  | no |  |
|  |  |  | yes |  |
|  |  |  | no |  |
|  |  |  | no |  |
|  |  |  | no |  |
|  |  |  | no |  |
|  |  |  | no |  |

Table 2: decision of transposing the context of the current block to be predicted and the prediction of this block, the value of , and the value of , and the neural network belonging to the neural network-based intra prediction mode used for prediction for each .

### Description of the neural networks belonging to the neural network-based intra prediction mode

|  |  |  |
| --- | --- | --- |
| **Network information (inference)** | | |
| Parameter number | block size | Parameter number |
|  |  |
| Parameter precision | 16-bit signed integer. | |
| MAC/pixel | block size | MACs/pixel |
|  |  |
| Total conv. layers |  | |
| Total FC layers | 4 cascaded fully-connected layers for the neural network predicting blocks.  3 cascaded fully-connected layers for the other six neural networks. | |
| Mem (MB) | 4.9 MB for the whole neural network-based intra prediction mode. | |
| Framework | SADL (C++ with a single thread) | |

### Combination of the neural network-based intra prediction mode and ISP

#### Restricted combination

In the proposed combination between the neural network-based intra prediction mode and ISP, ISP is allowed in luma exclusively and the same split types as in the original ISP are employed, i.e. horizontal and vertical splits. However, for a given luma CB predicted by the neural network-based intra prediction mode, ISP is ruled by four additional constraints compared to the constraints specific to the original ISP.

1. The luma CB can only be split into 4 equal-sized sub-partitions for both prediction and {transform coding, entropy coding of quantized transform coefficients}. For instance, a 8x8 luma CB predicted via the neural network-based intra prediction mode cannot be split vertically into 2 4x8 sub-partitions for prediction and 4 2x8 sub-partitions for {transform coding, entropy coding of quantized transform coefficients}.
2. A sub-partition of the luma CB cannot have a dimension strictly smaller than 4. For instance, a 4x16 luma CB predicted via the neural network-based intra prediction mode cannot be split vertically via ISP into 4 1x16 sub-partitions.
3. At least one of the two dimensions of the luma CB must be equal to 4.
4. The luma CB must belong to a CU in separate tree.

To summarize 1), 2), 3), and 4), for a given luma CB predicted by the neural network-based intra prediction mode, ISP is allowed if is true.

height of the luma CB.

: width of the luma CB.

: true if the luma CB is part of a CU in separate tree.

#### Additional combination with LFNST

For a given luma CB predicted by a non-neural network-based intra prediction mode, when ISP and LFNST are used, the LFNST implicit signaling is determined by the index of the intra prediction mode, and all the luma TBs arising from the ISP split share the same LFNST implicit signaling. In contrast, for a given luma CB predicted via the neural network-based intra prediction mode, when ISP and LFNST are used, for each luma TB independently, the neural network infers the LFNST implicit signaling for this luma TB while predicting it.

## Small ad-hoc deep learning (SADL) library

SADL (Small Ad-hoc Deep-Learning Library) is a header only small library for inference of neural networks. SADL provides both floating-point-based and integer-based inference capabilities. The inference of neural networks in NNVC is based on the SADL.

The table below summarizes the framework characteristics.

Table 4. Characteristics of SADL

|  |  |
| --- | --- |
| Language | Pure C++, header only. |
| Footprint | ~8k LOC, library ~300kB, no dependency |
| Optimization | Some SIMD at hot spots, e.g. convolution (conv2D) and automatic sparse matrix-vector multiplication |
| Compatibility | Onnx to SADL converter |
| Layer Supports | constants, add, maxPool, matMul (dense and sparse), reshape, ReLU, conv2D (strided, grouped, separated), mul, concat, max, leakyReLU, shape, expand, PReLU, flatten, transpose, Cond2DTranspose, Slicing, ScatterND, GridSample, Resize, Compare, Where, ReduceMean |
| Layer supports experimental | BatchNorm, Sigmoid |
| Type support | float, int32, int16, int8 |
| Quantization | Support adaptive quantizer per layer |
| License | BSD 3-Clause |

NNVC repository uses SADL as a submodule, pointing to the repository here: <https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/sadl>.

Documentation is available in the doc directory of the repository.

## Content-adaptive neural networks loop-filter

### Neural network

The content adaptation of the loop-filter is applied to the LOP3 loop-filter. That means the architecture is the same as the LOP3 loop-filter plus some multiplier parameters in the luma and chroma branches.

The multipliers are applied after the bias is added to the convolution result:

The inputs to the NN are the same as for the LOP3 filter, except for 2 additional flags that are used to turn on/off the multipliers if needed for certain frames. The multipliers are activated when the value of both flags is equal. The second flag is always 1. For the first flag, 1 indicates that the multipliers are enabled and 0 indicates that the multipliers are disabled.

### Content adaptation

The content adaptation is done at the encoder-side for each sequence, sequence QP and random-access segment. Given a certain sequence, sequence QP and random-access segment, the result of this process is a weight-update (difference between base multipliers and over-fitted multipliers) coded with the Neural Network compression and Representation (NNR) standard and carried within a Neural Network Filter Update (NNFU) APS in the first B-frame of the segment.

### Inference details

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | CPU only |
| Framework: | SADL |
| Number of GPUs per Task |  |
|  |  |
| Number of Parameters (Each Model) | 209918 |
| Total Number of Parameters (All Models) | 209918 |
| Parameter Precision (Bits) | Int16 |
| Memory Parameter (MB) | 0.4 |
| Multiply Accumulate (kMAC/pixel) | 17.3 |
| Calculation Method | on a block basis |
| Optional |  |  |
| Total Conv. Layers |  |
| Total FC Layers |  |
| Total Memory (MB) |  |
| Batch size: |  |
| Patch size | 128x128 / 256x256 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage (Total) |  |
| Peak Memory Usage (per Model) |  |
| Border handling |  |
| Other information: |  |
|  |  |

## Encoder

The content adaptation is done per RA segment independently and once the model is ready to be deployed, the encoding of the RA segment is also done independently from other segments. Adjacent RA segments have overlapping I-frames and these are filtered using the original LOP3 filter.

After a RA segment of a sequence-QP pair has been encoded using the respective adaptive LOP3 filter, a decision is performed to determine whether this segment or the one encoded with the anchor configuration (i.e., with the base model) is better in terms of delta PSNR. Therefore, a final merged bitstream may include some RA segments filtered with the original (base) LOP3 filter and some RA segments filtered with the adaptive LOP3 filter.

## Decoder

When the decoder parses an NNFU APS, the reconstruction of the overfitted LOP3 filter is triggered. The process relies on the original (base) LOP3 filter and the decoded NNR bitstream carried within the NNFU APS.

## Other loop filters

Content-adaptability is also supported for VLOP and LOP2 loop-filters.

## Content-adaptive neural network post-filter

### Neural network

Figure 6 shows the architecture of the NN. The input consists of the reconstructed luma and chroma samples, as well as the strength control value SliceQPY.

The architecture also includes multiplier parameters, which are applied after the bias is added to the convolution result:

where is the kernel, is the convolution operator, is the input, is the bias, is the multiplier and is the activation function.

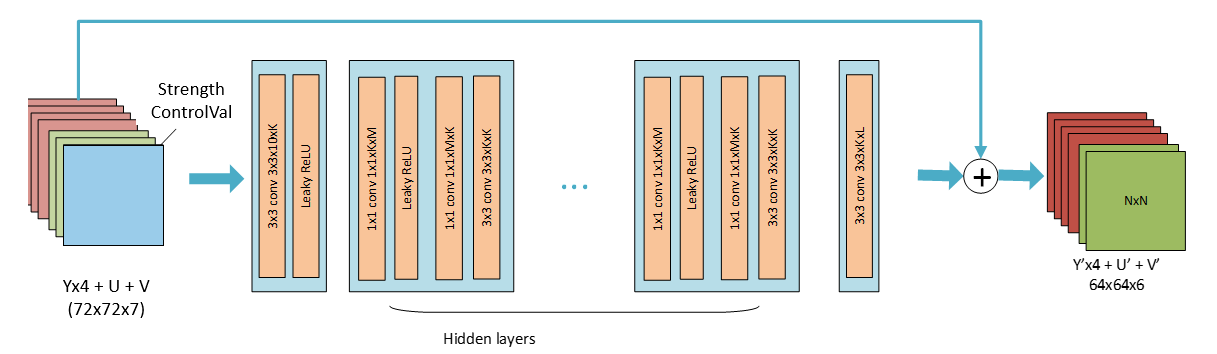


Figure . Architecture of the content-adaptive post-filter

Since the luma and chroma components have different resolutions, the input luma component is split into 4 channels following the pixel unshuffled operation. Similarly, the final output luma component is brought back to the real resolution by applying the pixel shuffle operation.

### Content adaptation

The content-adaptive post-filter includes four base filters trained offline. The content adaptation is achieved by over-fitting one of them on the test content. The over-fitting is done at the encoder side and only for the multiplier parameters. To recreate the over-fitted model at the decoder end, the resulting weight-update (difference between base multipliers and over-fitted multipliers) is coded with Neural Network compression and Representation (NNR) standard and sent within a Neural Network Post Filter Characteristics (NNPFC) SEI message.

### Inference details

Table III shows the network information in inference stage when using SADL int16 fixed-precision.

The content-adaptive post-filter includes the signalling of two NNPFC SEI messages (one with the characteristics of the base model and one with the NNR weight-update) and one Neural Network Post Filter Activation (NNPFA) SEI message. The latter activates/enables an NN post-filter for the whole video encoded video sequence.

Table . Network Information of the content-adaptive post-filter in inference stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | CPU only |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Number of Parameters (Each Model) | 109068 |
| Total Number of Parameters (All Models) | 436272 (x4) |
| Parameter Precision (Bits) | 16 |
| Memory Parameter (MB) | 0,832122803 |
| Multiply Accumulate (kMAC/pixel) | 34 (block) |
| Optional |  |  |
| Total Conv. Layers | 35 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 144x144 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage (Total) |  |
| Peak Memory Usage (per Model) |  |

## Neural network-based super resolution

### Neural network for luma component

The neural network architecture of the super resolution is illustrated in Figure 9. The chroma components are upsampled to the same size of luma components before feeding into the network. Then a 1×1 convolution is used to fuse the concatenated features. Split of luma and chroma with different complexity is designed that the chroma branch includes a smaller number of channels and basic backbone blocks compared to luma branch. Scaling ratio (*Sr*) is set as an input to the model:

* For scaling ratio is 2.0x, *Sr =* 1;
* For scaling ratio is 1.5x, *Sr =* 2;
* For scaling ratio is 1.0x, *Sr =* 0.

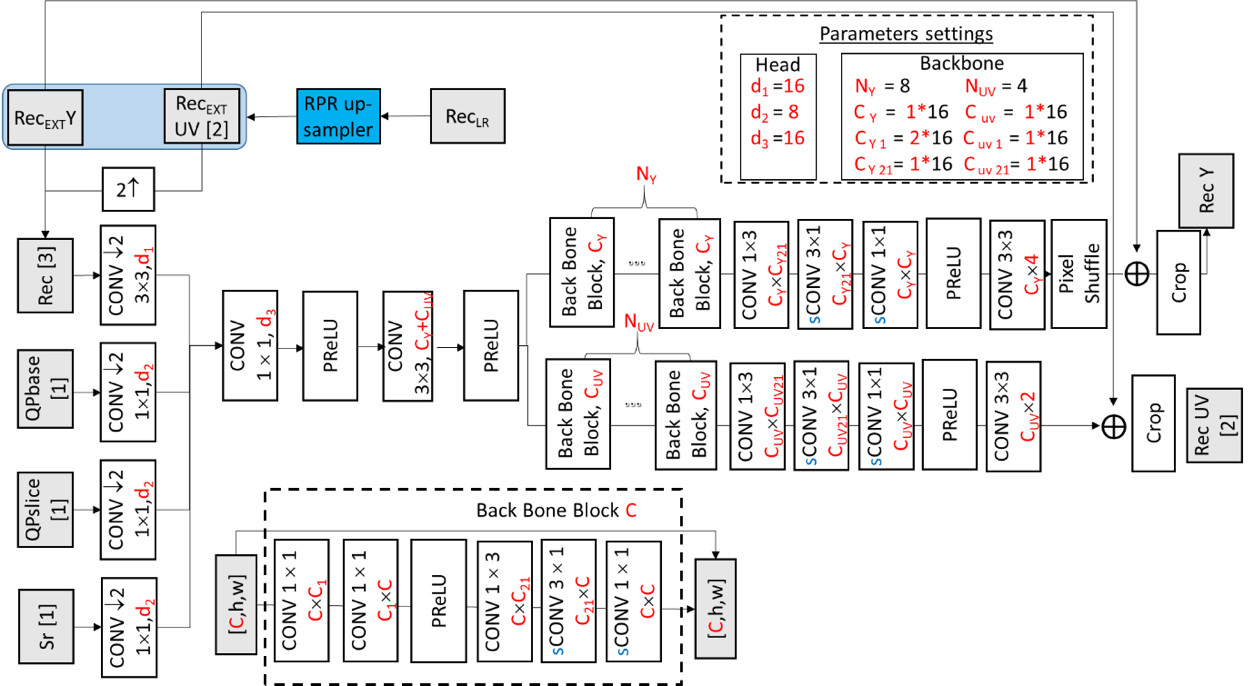


Figure Architecture of the super resolution

### GOP level encoding resolution decision

The encoder enables resampling-based method only if . One of resampling scale ratios {×1.0 (original size), ×2.0 (half size) and ×1.5 (2/3 size)} is selected at GOP level and the same scale ratio is applied on all frames in a common one GOP. To determine the resampling ratio, the PSNR value and initial QP for the first frame of GOP are exploited. Specifically, the first frame of GOP is downscaled to quarter resolution and then resampled to the original resolution by using the existing RPR technology. The PSNR is calculated between the original frame and the down-up scaled frame. Considering the different characteristics between luma and chroma components, PSNRs from different components are calculated in the scale ratio decision. The scale ratio decision process is shown in Figure 10.

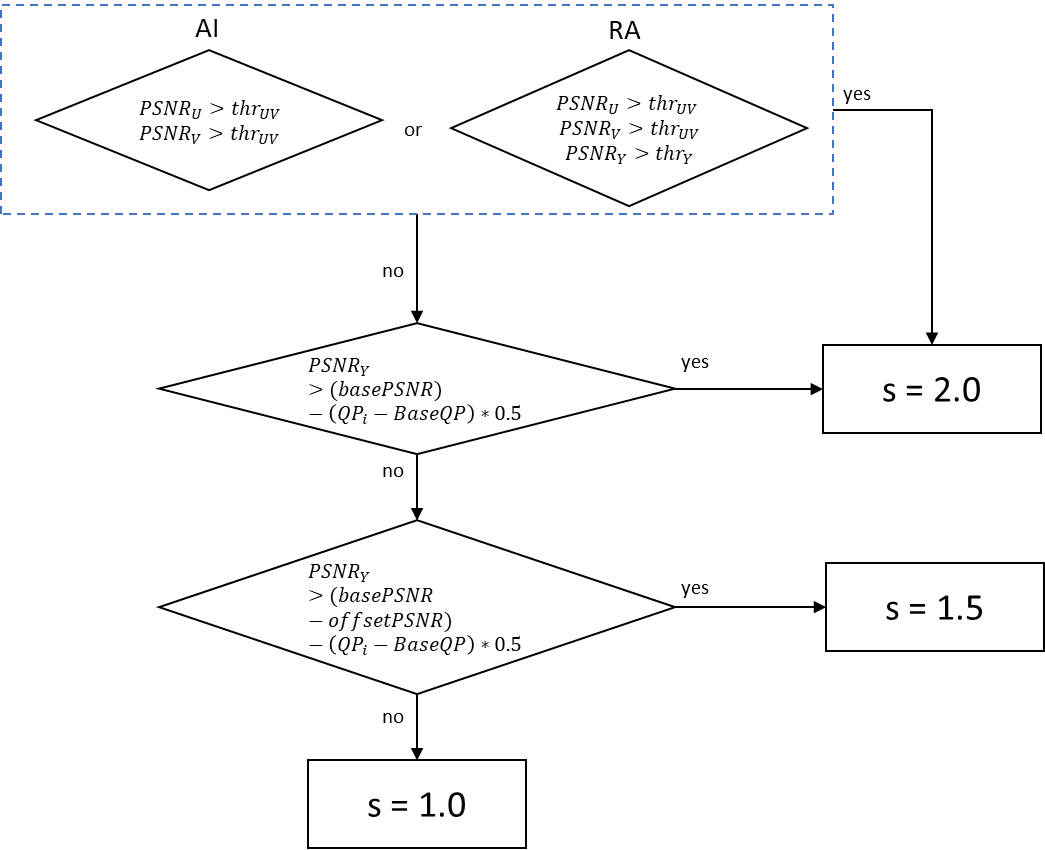


Figure GOP level encoding resolution decision.

* For RA, if and , and , scaling ratio is set equal to 2.0.

For AI, if and , scaling ratio is set equal to 2.0.

* else
  + if , scaling ratio is set equal to 2.0.
  + else if , scaling ratio is set equal to 1.5.
  + else, scaling ratio is set equal to 1.0.

where , and represent down-up scaled luma and chroma PSNR for the first frame of GOP, respectively. *basePSNR* and represent predefined threshold PSNR. is set equal to different values for RA and AI, respectively. According to selected scaling ratio for *i*-th GOP, QP offset is set for ×2.0 and ×1.5 respectively.

### Inference details

SADL (see Section 1.3) is used for performing the inference of the CNN filters. Both floating point-based and fixed point-based implementations are supported. In the fixed-point implementation, both weights and feature maps are represented with int16 precision using a static quantization method. The network information in the inference stage is provided in Table 6.

Table Network Information of super resolution in Inference Stage

|  |  |  |
| --- | --- | --- |
| Network Information in Inference Stage | | |
| Mandatory | HW environment: | Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
| Number of Parameters (Each Model) | 19.6 K |
| Total Parameter Number | 19.6 K |
| Parameter Precision (Bits) | Int 16 |
| Memory Parameter (MB) | 39.2 KB |
| Multiply Accumulate (MAC) | 4.7 kMAC/pixel |

# Training description of Neural Network-based Video Coding Software

The training of NN-based tool typical involve three steps: compressing a dataset and dumping necessary training data, loading training data, training using the loaded training data. The first two steps are usually based on the data dumper and data loader in NNVC, while the third step relies on the training method designed for each specific tool.

A high-level description of training methods for NN-based tools is provided below. To reproduce the training of a specific tool, it is recommended to refer to the training scripts in the NNVC software.

## Low operating point neural network-based loop filter

The following describes the three training stages of LOP2. LOP3 is trained with only Stage 3 using the LOP2 dataset to allow a direct comparison to LOP2.

Three stages of LOP2:

* Stage 1:
  + Images from DIV2K dataset are used. The images are compressed using AI configuration. NNVC SW without NN tools is used for encoding and decoding.
  + A LOP model is trained using the extracted DIV2K dataset.
* Stage 2:
  + Sequences from BVIDVC dataset and TVD dataset are used. The sequences are compressed using RA configuration. NNVC SW with the LOP model from stage 1 is used for encoding and decoding.
  + A stage 2 LOP model is trained using the extracted DIV2K dataset from stage 1 and the extracted BVIDVC and TVD datasets from stage 2.
* Stage 3:
  + Sequences from BVIDVC dataset and TVD dataset are used again. The sequences are compressed using RA configuration. NNVC SW with the LOP model from stage 2 is used for encoding and decoding.
  + A stage 3 LOP model is trained using the extracted DIV2K dataset from stage 1 and the extracted BVIDVC and TVD datasets from stage 3.

## Very Low operating point neural network-based loop filter set

VLOP2 model available in the reference NNVC SW has been trained using Stage 3 LOP2 dataset following the procedure described in Section 5.1. Details of the training hyper parameters and training strategy are presented in JVET-AI0107.

## Neural network-based intra prediction

The workflow of the iterative training of the neural networks belonging to the neural network-based intra prediction mode is displayed in Figure 9.

* At cycle 0, VTM-11-NNVC without the neural network-based intra prediction mode and without any Filter-Set extracts the data for training the neural networks. Then, the 7 neural networks are trained, initializing their parameters randomly.
* At cycle 1, VTM-11-NNVC with the neural network-based intra prediction mode using the parameters trained at cycle 0 and without any Filter-Set extracts the data for training the neural networks. Then, the 7 neural networks are trained, initializing their parameters from their state at the end of cycle 0.
* At cycle 2, VTM-11-NNVC with the neural network-based intra prediction mode using the parameters trained at cycle 1 and without any Filter-Set extracts the data for training the neural networks. Then, the 7 neural networks are trained, initializing their parameters from their state at the end of cycle 1. Then, using the same training data, the trainings of these 7 neural networks are resumed, introducing this time a sparsity constraint on their weights.
* At cycle 3, VTM-11-NNVC with the neural network-based intra prediction mode using the parameters trained at cycle 2 and without any Filter-Set extracts the data for training the neural networks. Then, the transform prediction part of each of the 7 neural networks is trained, initializing their parameters from their state at the end of cycle 2.

Diagram

Description automatically generated

Figure : workflow of the iterative of the neural networks belonging to the neural network-based intra prediction mode.

## Content-adaptive neural network post-filter

The training of the content-adaptive neural network post-filter consists of two stages as follows,

Stage 1: generation of four base models.

Stage 2: over-fitting of one base model for each test sequence and QP point, followed by weight-update coding with NNC.

## Content-adaptive neural network loop-filter

The base model is using LOP2 training procedure. Content adaptation is described in model description.

## Neural network-based super resolution

The training of neural network-based super resolution is straightforward. The training can be done only from stage 3. The NNVC software with RPR enabled is used to generate training datasets.

## High Operating Point model training

HOP4 model available in the reference NNVC SW has been trained using Stage 3 HOP dataset following the procedure described below.

The detailed training description is available in the readme file of the HOP training directory.

### Overview

A screenshot of a computer program

Description automatically generated

Figure 13 Training steps for HOP model creation

The figure above describes the 3 stages needed to fully train the HOP model:

* Stage I:
  + Extract a dataset of intra coded frames using VTM
  + Train HOP on the dataset: resulting model is HOP.I
* Stage II:
  + Extract a dataset of frames using VTM and HOP model from stage I
  + Train HOP on the dataset: resulting model is HOP.II – Integerized the model.
* Stage III:
  + Extract a dataset of frames using VTM and HOP model from stage II
  + Train HOP on the dataset: resulting model is HOP.III (final model)

Details for training can be found in training description document JVET-AE0191 [7].

All official models can be found in the NNVC repository at <https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM>.

### Training characteristics

Network Information for NN-based Video Coding Tool Testing in Training Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Training Stage** | | |
| Mandatory | GPU Type | GPU: Tesla-V100-SXM2-32GB/ Tesla-A100-40GB – used memory: 25GB |
| Framework: | PyTorch v1.9 |
| Number of GPUs per Task | 1 |
|  |  |
| Epoch: | Stage 1: 40, stage 2: 20 (19), stage 3: 20 |
| Batch size: | 64 |
| Training time: | 6-12h/epoch |
| Training data information: | DIV2K, BVI-DVC, TVD |
| Training configurations for generating compressed training data (if different to VTM CTC): | See below |
|  | Loss function: | L1, L2 |
| Optional |  |  |
| Number of iterations | 101k/epochs |
| Patch size | 144144 |
| Learning rate: | 1e-4 |
| Optimizer: | ADAM |
| Preprocessing: | Data augmentation (offline stage 1, online stage 2 and 3) |
| Other information: |  |

### Prerequisite

* Dataset to use described in Common Test Conditions document JVET-AD2016 [11]
  + Div2k, BVI, TVD
  + Use original naming, directories.
* Storage preparation:
  + Needed space can be found in the readme.md
* MD5sum:
  + MD5sum: can be found on the wiki of the repository at [wiki](https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM/-/wikis/MD5-files)
  + json files contains md5sum whenever possible
* Repository: <https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM>
* Base for HOP training scripts inside the repository: [training/training\_scripts/NN\_Filtering\_HOP](https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM/-/blob/VTM-11.0_nnvc/training/training_scripts/NN_Filtering_HOP/readme.md)
* Preparation:
  + 1 central file to edit everything related to your configuration: paths.json
  + Other parameters are only to adapt your training strategy etc.

#### Training from scratch

All steps are described in [readme.md](https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM/-/blob/VTM-11.0_nnvc/training/training_scripts/NN_Filtering_HOP/readme.md).

#### Partial training

The training can be done only from stage 3. In this case, the int16 model from stage 2 is used to generate the dataset of stage III.

MD5sums of stage 3 datasets and encoded files can be found in <https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/nnvc-ctc>/HOP/md5

The md5sums of datasets (to be checked in the json file) are:

|  |  |  |
| --- | --- | --- |
| Dataset name | Number of patches | Md5sum |
| TVD | 432000 | "6468995aa583ca941af37fea2ddac20c" |
| TVD valid | 48000 | "0fdedc21e41496d4694a2b05dbcb7a5f" |
| BVI | 2157210 | "9162efd535a6c65f6bd311e405694d0a" |
| BVI valid | 97790 | "e7231f428e54ed461cff3628af70e241" |

The md5sums of individual bitstream for each dataset can be found in the repository above to detect possible issue early.

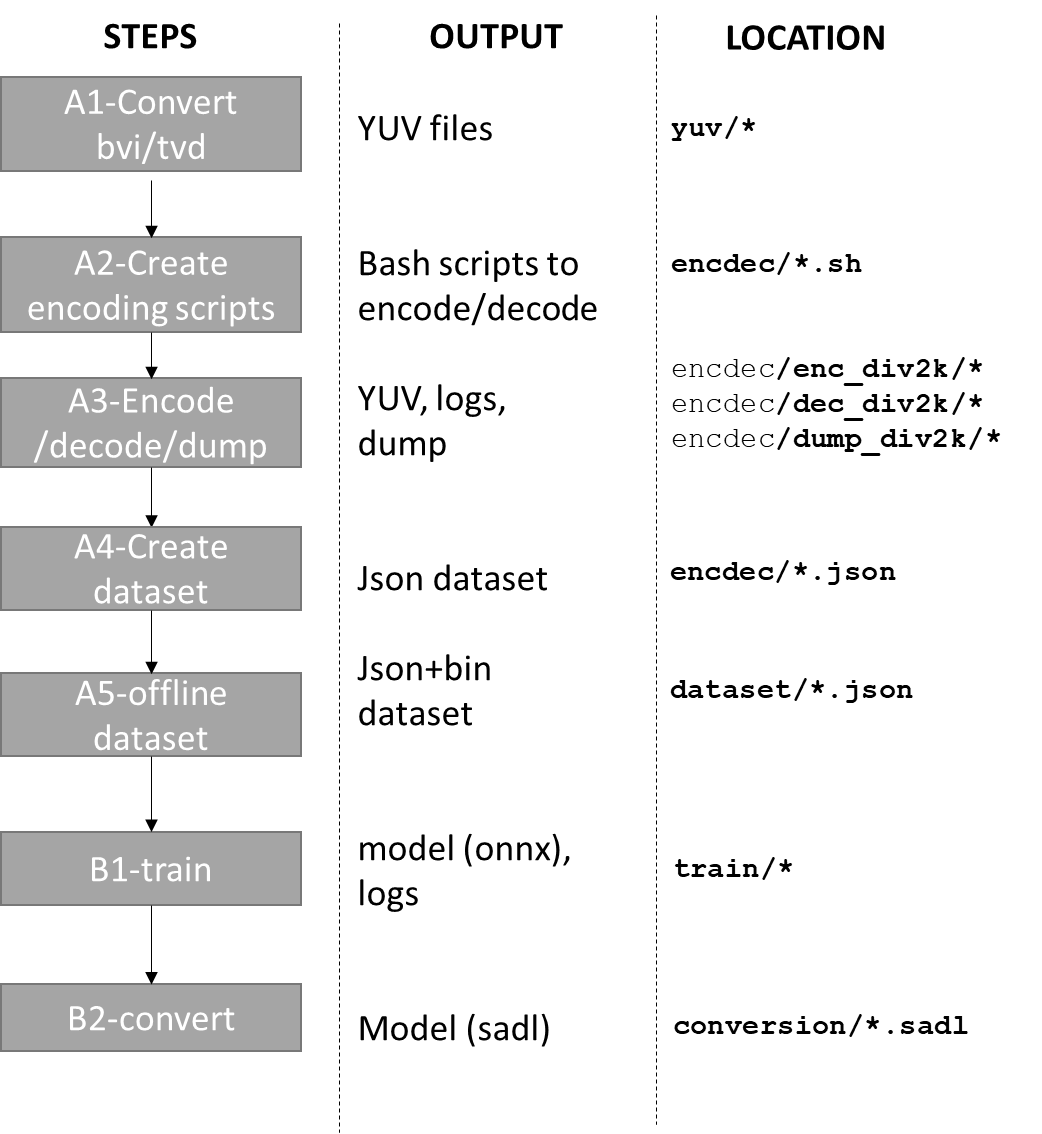
### Training Model stage I

A screenshot of a computer program

Description automatically generated

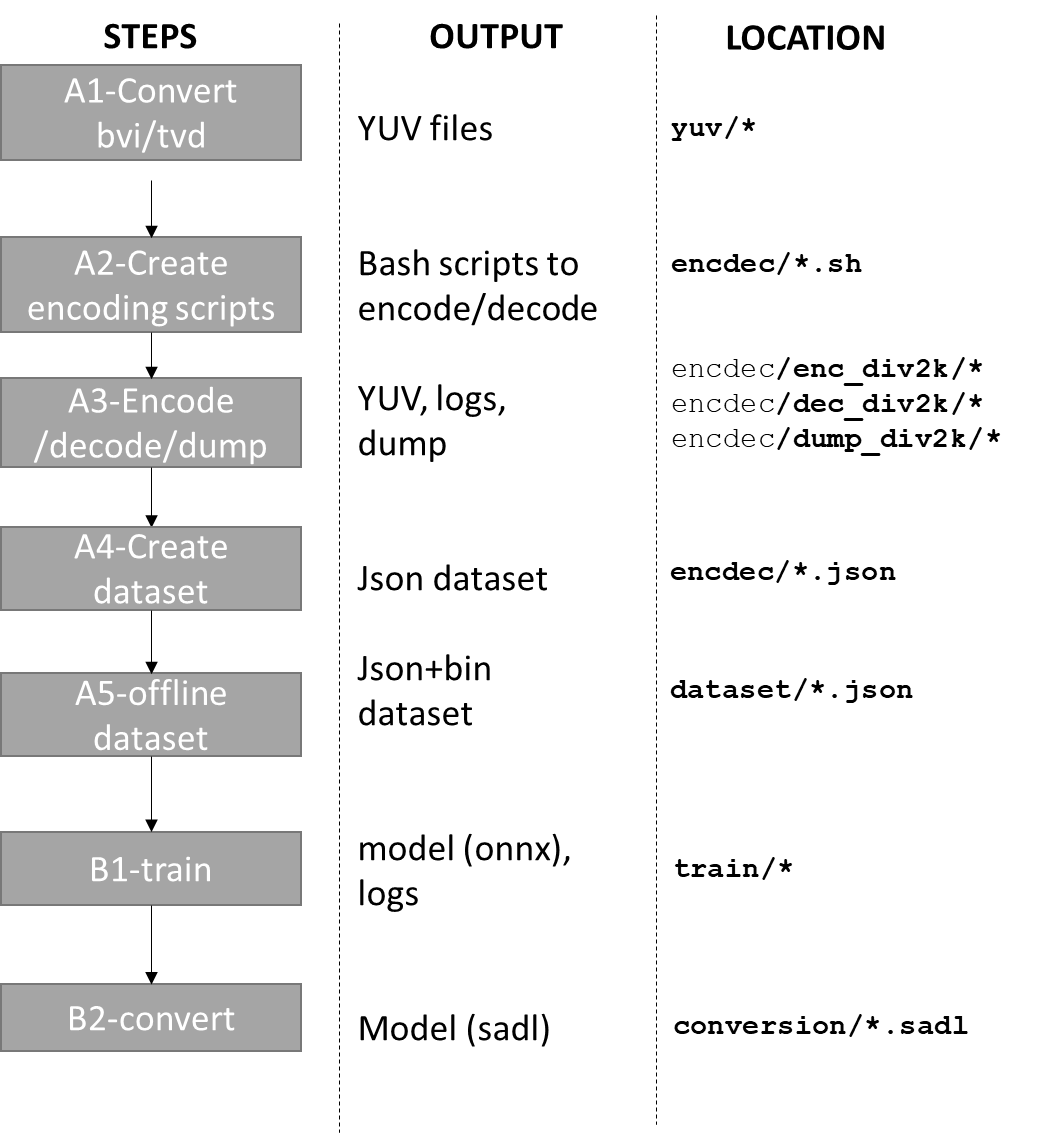
* 1. Data extraction for intra from vanilla VTM
     1. encoder turns on md5sum, just NNVC-5.0 all NNVC tools disabled, encoder\_intra\_vtm.cfg, just first frame
     2. QPs = 22,27,32,37,42 + 19,24,29,34,39
     3. DIV2K (4:4:4 RGB 🡪 YUV420 10 bits, version ffmpeg to be specified,
     4. Rotation, flipping, down-sampling (list of output + md5sum)
     5. Reconstruction is extracted before de-block
     6. All CTUs are extracted (except those on the border)
  2. Model stage I training from scratch for Intra data:
     1. Random initialization
     2. Use all CTUs
     3. Luma/chroma balance: 6:1:1
     4. Learning Rate: 0.0001, batch size: 64, scheduler: 0.1 decay at epoch 30 and 34
     5. Norm switch: L1 up to epoch 37, then L2
     6. 40 epochs total

### Training Model stage II



1. Data extraction with model#1:
   * 1. encoder turns on md5sum, NNVC-5.0 + Model stage I for I-frames only, encoder\_ra\_model1.cfg, 65 frames (last is duplicated), Intra Period 64, GOP=32.
     2. QPs = 22,27,32,37,42
     3. List of sequences
        1. TVD: 65 frames, 20 sequences
        2. BVI (YUV420), 620 sequences
        3. All reconstructed CTU are extracted before de-block for a subset of frames + DVI2K training data as in training stage I
2. Model stage II training with extracted data:
   * 1. Random initialization
     2. Use all CTUs
     3. Rotation and flipping on the fly
     4. Luma/chroma balance: 6:1:1
     5. Learning Rate: 0.0001, batch size: 64, scheduler: 0.1 decay at epoch 15 and 17
     6. Norm switch: L1 up to epoch 18, then L2
     7. 19 epochs total

### Training stage III



1. Data extraction with Model stage II:
   * 1. encoder turns on md5sum, NNVC-5.0 + Model stage II (both Intra and Inter frames), encoder\_ra\_vtm.cfg + model stage 2, 65 frames (last is duplicated), Intra Period 64, GOP=32.
     2. QPs = 22,27,32,37,42
     3. Same sequences and frames as for stage 2
2. Model *Stage III* training with extracted data:
   * 1. Random initialization
     2. Use all CTUs
     3. Rotation and flipping (on the fly augmentation)
     4. Luma/chroma balance: 12:1:1
     5. Learning Rate: 0.0002, batch size:32, scheduler: 0.1 decay at epoch 15 and 17, 18
     6. Norm switch: L1 up to epoch 18, then L2
     7. 20 epochs total

# Legacy filters

## Neural network-based loop filter set 0

### Pre-processing and post-processing of chroma

In filter set 0, the filter with a single model is designed to process three components. Since the resolutions of luma and chroma are different, pre-processing and post-processing steps are introduced to up-sample and down-sample chroma components respectively as shown in Figure. 1. In the resampling process, the nearest-neighbor interpolation method is used.

A diagram of a filter

Description automatically generated

*Figure. 14 the pre-processing and post-processing units*

### Neural network

The network structure of the CNN filter is shown in Figure. 2. Along with the reconstructed image (rec\_yuv), additional side information is also fed into the network, such as the prediction image (pred\_yuv), slice QP, base QP and slice type. In the ResBlock, the number of channels firstly goes up before the activation layer, and then goes down after the activation layer. Specifically, K and M are set to 64 and 160 respectively, and the number of Resblock is set to 32.

图示

描述已自动生成

*Figure. 2*. *Architecture of the CNN in filter set 0.*

### Combination with conventional filters

A black background with a white square

Description automatically generated

*Figure. 3*. *Implementation of the CNN in filter set 0.*

As shown in Figure.3, the reconstructed samples before DBK are fed into the CNN based filter (CNNLF), then final filtered samples are generated by blending the result of CNNLF and SAO. This blending process can be briefly formulated as:

There are four candidates, 1, 0.75, 0.5 and an adaptive weight, for the blending weight. With regard to the adaptive weight, its derivation is based on least square method. If the adaptive weight is selected, the blending weight is signaled for each color component in the slice header.

### Mode selection

The CNN filter can be turned on/off at the CTU level and slice level. For each enabling type, there are four blending ways. Therefore, there are nine modes to be evaluated by RDO at encoder. The final selected mode would be signaled in the slice header.

Table 1. Parameter selection of filter set 0

|  |  |  |
| --- | --- | --- |
| Mode | On/off type | Blending weight (w) |
| 0 | Disable at slice level | None |
| 1 | Enable at slice level | Adaptive weight |
| 2 | 1 |
| 3 | 0.75 |
| 4 | 0.5 |
| 5 | Enable at CTU level | Adaptive weight |
| 6 | 1 |
| 7 | 0.75 |
| 8 | 0.5 |

### Base QP adjustment

Base QP is fed into the CNN filter as shown in Figure. 2. To improve adaptation, an offset can be added to the base QP (the adjusted base QP is used as the input to the NN filter) at slice level. The offset candidates are {-5, 5}. For example given the offset -5, the actual input base QP to the filter becomes (BaseQP - 5) for the current slice.

**Encoder approach**

The proposed encoder only filters one out of every four CTUs during the process of selecting the best base QP offset to save encoding time. As shown in Figure 4, only shaded CTUs are considered for calculating distortions of using different BaseQP candidates {BaseQP, BaseQP-5, BaseQP+5}. After the candidate with the smallest cost is selected, the encoder filters the rest of CTUs (non-shaded ones in Figure. 4) by appying the best offset to the base QP.

图片包含 背景图案

描述已自动生成

*Figure. 4. Encoder optimization 2.*

### Encoder-only Optimization

To more accurately estimate the rate-distortion (RD) cost with integrated NN-based in-loop filters, an encoder-only NN filter is involved in the partitioning decision process. In the partitioning mode decision, the distortion between NN filtered samples and original samples is calculated, and then the optimal partitioning mode is selected based on calculated distortion to make the partitioning decision more accurate. To reduce complexity, only few ResBlocks (see Section 3.1.2) are used in the network structure. The NN filter in the RDO process is implemented with SADL using int16 precision. This encoder-only NN tool is disabled by default.

### Inference details

SADL (see Section 3.5) is used for performing the inference of the CNN filters. Both floating point-based and fixed point-based implementations are supported. In the fixed-point implementation, both weights and feature maps are represented with int16 precision using a static quantization method. The network information in the inference stage is provided in Table 2.

Table 2. Network Information of filter set 0 in Inference Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Number of Parameters (Each Model) | 1.9M |
| Total Parameter Number | 1.9M, one model in total |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 7.6MB, one model in total  int: 3.8MB, one model in total |
| Multiply Accumulate (kMAC/pixel) | 485 (assuming frame-level input)  615 (assuming block-level input) |
| Optional |  |  |
| Total Conv. Layers | 101 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 144144 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

## Neural network-based loop filter set 1

### Neural network for luma component

There are two regular networks in filter set 1, one for luma and one for chroma.

The inputs of the luma network comprise the reconstructed luma samples (rec), the prediction luma samples (pred), boundary strengths (bs), QP, and the block type (IPB). The numbers of feature maps and residual blocks are set as 96 and 8 respectively. The structure of the luma network is depicted in Figure 5 (a) – (c).

A screenshot of a computer

Description automatically generated

*(a) Head of luma network. The inputs are combined to form the input y to the next part of the network.*

A black rectangular object with text

Description automatically generated

*(b) The k-th residual block (k=0..7). The output y of the head is fed into a first residual block with input z0=y. The output z1 is then fed into another such residual block.*

A diagram of a computer

Description automatically generated

*(c) The output of the last residual block is fed into this last part of the network.*

*Figure 5*. *Architecture of the CNN in filter set 1.*

### Neural network for chroma component

Luma information is taken as additional input for the in-loop filtering of chroma. Considering the resolution of luma is higher than chroma in YUV 4:2:0 format, features are first extracted separately from luma and chroma. Then luma features are down-sampled and concatenated with chroma features. The inputs of the chroma network include reconstructed luma samples (recY), reconstructed chroma samples (recUV), predicted chroma samples (predUV), boundary strength (bsUV), and QP. Regarding network backbone, chroma components use the same one as luma.

### Temporal filter

Filter set 1 contains an additional in-loop filter, namely temporal fitter, which takes collocated blocks from the first picture in both reference picture lists to improve performance. The two collocated blocks are directly concatenated and fed into the network as shown in Figure 6. When enabling temporal filtering feature, the temporal filter is applied to the luma component of pictures in three highest temporal layers, while the regular luma and chroma filters are used for other cases. By default, this temporal filtering feature is disabled.



*Figure 6*. *Temporal in-loop filter. Only head part is illustrated, other parts remain the same as in Figure 5 (b)-(c). {Col 0, Col 1} refers to collocated samples from the first picture in both reference picture lists.*

### Adaptive inference granularity

The granularity of the filter determination and the parameter selection is dependent on resolution and QP. Given a higher resolution and a larger QP, the determination and selection will be performed in a larger region.

### Parameter selection

Each slice or block could determine whether to apply the CNN-based filter or not. When the CNN-based filter is determined to be applied to a slice/block, which conditional parameter from a candidate list including two candidates derived from QP could be further decided. Denote the sequence/slice level QP as q (inter slice and intra slice use slice QP and sequence QP respectively), the candidate list includes conditional parameters {Param\_1, Param\_2}. For low temporal layers, Param\_1 = q, Param\_2 = q5. For high temporal layers, Param\_1 = q, Param\_2 = q5. In other words, the second candidate is different across different temporal layers.

The selection process is based on the rate-distortion cost at the encoder side. Indication of on/off control as well as the conditional parameter index, if needed, are signalled in the bitstream. Figure. 6 shows the diagram of parameter selection at encoder and decoder sides. All blocks in the current frame need to be processed with all conditional parameters first. Then all costs, i.e. Cost\_0, ..., Cost\_N+1, are calculated and compared against each other to achieve optimum rate-distortion performance. In Cost\_0, CNN-based filter is prohibited for all blocks. In Cost\_i, {i = 1, 2, 3, ..., N}, the parameter Param\_i is used for all blocks. In Cost\_N+1, different blocks may prefer different parameters, and the information regarding whether to use CNN-based filter or which parameter to be used is signaled for each block. At decoder side, whether to use CNN-based filter or which parameter to be used for a block is based on the Param\_Id parsed from the bit-stream as shown in Figure. 6 (b).

Note that for all-intra configuration, parameter selection is disabled while filter on/off control is still preserved. A shared conditional parameter is used for the two chroma components to ease the burden in worst case at decoder side. In addition, the max number of conditional parameter candidates, i.e. N, could be specified at encoder side (N = 2 by default).



*Figure. 6*. *(a)* *Parameter selection at encoder side. (b) Parameter selection at decoder side*.

### Residue scaling

When a NN filter is being applied to reconstructed pictures, a scaling factor is derived and signaled for each color component in the slice header. The derivation is based on least square method. The difference between the input samples and the NN filtered samples (residues) are scaled by the scaling factors before being added to input samples.

### Combination with deblocking filter

To enable a combination with deblocking, the input samples used in the residual scaling is the output of deblocking filtering. The residual scaling process is shown below, where and refer to the outputs of NN filtering and deblocking filtering respectively.

=

### Encoder-only optimization

Different from NNVC-2.0, EncDbOpt is also enabled for AI configuration.

For a better estimation of rate-distortion (RD) cost in the case the NN filter is used, the proposed encoder introduces NN-based filtering into the rate-distortion optimization (RDO) process of partitioning mode selection. Specifically, a refined distortion is calculated by comparing the NN filtered samples and the original samples. The partitioning mode with the smallest rate-refined distortion cost is selecte as the optimal one. To reduce complexity, several fast algorithms are applied. First, NN model is simplified by using a less number of residual blocks. Second, parameter selection is not allowed for the NN filtering in the RDO process Third, the proposed technique is only applied to the coding units with height and width no larger than 64. The NN filter used in the RDO process is also implemented with SADL using fixed point-based calculation. This NN-based encoder-only method is disabled by default.

### Inference details

SADL (see Section 3.5) is used for performing the inference of the CNN filters. Both floating point-based and fixed point-based implementations are supported. In the fixed-point implementation, both weights and feature maps are represented with int16 precision using a static quantization method. The network information in the inference stage is provided in Table 3.

Table 3. Network Information of filter set 1 in Inference Stage

|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Total Parameter Number | 1.55M/model, 2 models in total for all tests |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 6.2MB/model, 2 models  int: 3.1MB/model, 2 models |
| Multiply Accumulate (kMAC/pixel) | 532 (assuming frame-level input)  673 (assuming block-level input) |
| Optional |  |  |
| Total Conv. Layers | 25 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 144144, 272272 |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

## Training Legacy Neural network-based loop filter set 0

In order to effectively enhance the NN model generalization, an iterative training method is designed to better maintain consistency between the training process and the inference process. As shown in Figure 7, the proposed iterative training method contains the initial training stage and the iterative training stage. In the initial training stage, the training data is generated by the anchor configured with the common test conditions. In the iterative training stage, the NN model obtained from the previous training process is integrated into anchor and the training data is generated by the NN-filter based codec.

A diagram of a diagram

Description automatically generated

*Figure. 7. The Iterative training method*

In the initial training stage, NNVC (--NnlfOption=0) is used to compress all training images under all-intra and random-access configuration, respectively. The reconstructed images together with additional side information are generated and utilized to train the NN filter.

In the iterative training stage, NNVC (--NnlfOption=1) with the integrated NN model from previous training stage is used to compress all training images under random-access configuration and the training data for I slices is still used from initial training stage. Theoretically, the more times of the iterative training stages, the better performance. However, only up to two times of training stages including the initial training stage are used.

In addition to the above real-iterative training method, pseudo-iterative method training can also be used. That’s to say, the training data used for each training stage can be generated by the codec whose performance is comparable with the codec enhanced by the latest training model, then the initial training stage can be skipped.

## Training Neural network-based loop filter set 1

### Regular filters

To effectively train the luma and chroma NN models, an iteratively conducted two-stage training is adopted to better align the settings during training and testing as shown in Figure 8. In the first stage, the training data is generated by VTM under AI and RA configurations and separate intra and inter models are trained. In the second stage, the models are integrated into VTM to generate the training data under random access configuration. To train the combined intra and inter models, the training data uses stage I AI data and stage II RA data.

A black background with a black square

Description automatically generated with medium confidence

*Figure 8. Iterative training with two stages.*

In training stage I, NNVC (--NnlfOption=0) is used to compress training images under all-intra and random access configurations. The reconstructed images together with other auxiliary information are collected and utilized for training intra frame filters and inter frame filters.

In training stage II, NNVC (--NnlfOption=2) equipped with the models from the previous training stage is used to compress training videos under random-access setting. That is to say, the intra frames and the inter frames will be processed by the intra filters and inter filters obtained in training stage I, respectively. Then, the AI data from stage I and RA data from stage II are used as training data to train the combined intra and inter models.

### Temporal filter

Training of temporal filter involves one step. NNVC (--NnlfOption=2) is used to compress training videos under random access configuration. The reconstructed images together with other auxiliary information are collected and utilized for training the temporal filter.

# References

1. B. Bross, J. Chen, S. Liu, and Y.-K. Wang "Versatile Video Coding Editorial Refinements on Draft 10," document JVET-T2001, 20th JVET meeting: by teleconference, 7 October – 16 October 2020.
2. J. Chen, Y. Ye, S. Kim "Algorithm description for Versatile Video Coding and Test Model 11 (VTM 11) ," document JVET-T2002, 20th JVET meeting, by teleconference, 7 October - 16 October 2020.
3. <https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/sadl>.
4. <https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tree/VTM-11.0>.
5. E. Alshina, F. Galpin, “BoG report on NN-filter design unification”, document [JVET-AD0380](https://jvet-experts.org/doc_end_user/current_document.php?id=12944), 30th Meeting, Antalya, TR, 21–28 April 2023
6. E. Alshina, F. Galpin, “AhG14 & AHG11: Report on AhG teleconference on high operation point (HOP) unified filter training “, [JVET-AE0042](https://jvet-experts.org/doc_end_user/current_document.php?id=12990), 31st Meeting, Geneva, CH, 11–19 July 2023
7. F. Galpin, S. Eadie, D. Rusanovskyy, Y. Li, Y. Li, J. Li, L. Wang, R. Chang, Z. Xie, E. Alshina “AhG11: EE1-0 High Operation Point model”, [JVET-AE0191](https://jvet-experts.org/doc_end_user/current_document.php?id=13154), 31st Meeting, Geneva, CH, 11–19 July 2023
8. F. Galpin, S. Eadie, D. Rusanovskyy, Y. Li, Y. Li, J. Li, L. Wang, R. Chang, Z. Xie, E. , “AhG11: HOP training process and models”, [JVET-AE0289](https://jvet-experts.org/doc_end_user/current_document.php?id=13252), 31st Meeting, Geneva, CH, 11–19 July 2023
9. D. Rusanovskyy, F.Galpin “AhG11: Performance of the NNVC HOP with quantized Stage II model”, [JVET-AE0291](https://jvet-experts.org/doc_end_user/current_document.php?id=13254), 31st Meeting, Geneva, CH, 11–19 July 2023
10. F. Galpin, D. Rusanovskyy, Y. Li, L. Wang, Z. Xie, Y.Li, R. Chang, J. Li, E. Alshina “AhG11: HOP full results”, [JVET-AF0041](https://jvet-experts.org/doc_end_user/current_document.php?id=13285), 32nd Meeting, Hannover, DE, 13–20 October 2023
11. E. Alshina, R.-L. Liao, S. Liu, A. Segall, “Common test conditions and evaluation procedures for neural network-based video coding technology”, JVET-AD2016, 30th Meeting, Antalya, TR, 21–28 April 2023