|  |
| --- |
| **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 231** |
| **Geneva, CH, 11–19 July 2023** |
| |  |  | | --- | --- | | **Title:** | **Description of algorithms and software in neural network-based video coding (NNVC) version 4** | | **Source:** | **Convenor (Jens-Rainer Ohm)** | | **Type:** | **General** | | **Subtype:** | **N/A** | | **Status:** | **Approved** | | **Date:** | **2023-09-04** | | **Expected Action:** | **Info** | | **Action due date:** | **N/A** | | **No. of pages** | **33** (without 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** | |

|  |  |
| --- | --- |
| **文本  中度可信度描述已自动生成徽标  描述已自动生成Joint Video Experts Team (JVET)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29**  31st Meeting, Geneva, CH, 11–19 July 2023 | Document: JVET-AE2019-v2 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Algorithm Description for Neural Network-based Video Coding (NNVC-6.0)** | | |
| *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 Neural Network-based Video Coding (NNVC 6) software algorithm description. 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](#_Toc144730166)

[1 Introduction 3](#_Toc144730167)

[2 Scope 3](#_Toc144730168)

[3 Algorithm description of Neural Network-based Video Coding Software 4](#_Toc144730169)

[3.1 Neural network-based loop filter set 0 4](#_Toc144730170)

[3.1.1 Pre-processing and post-processing of chroma 4](#_Toc144730171)

[3.1.2 Neural network 4](#_Toc144730172)

[3.1.3 Combination with conventional filters 5](#_Toc144730173)

[3.1.4 Mode selection 5](#_Toc144730174)

[3.1.5 Base QP adjustment 5](#_Toc144730175)

[3.1.6 Encoder-only Optimization 6](#_Toc144730176)

[3.1.7 Inference details 6](#_Toc144730177)

[3.2 Neural network-based loop filter set 1 7](#_Toc144730178)

[3.2.1 Neural network for luma component 7](#_Toc144730179)

[3.2.2 Neural network for chroma component 7](#_Toc144730180)

[3.2.3 Temporal filter 8](#_Toc144730181)

[3.2.4 Adaptive inference granularity 8](#_Toc144730182)

[3.2.5 Parameter selection 8](#_Toc144730183)

[3.2.6 Residue scaling 9](#_Toc144730184)

[3.2.7 Combination with deblocking filter 9](#_Toc144730185)

[3.2.8 Encoder-only optimization 9](#_Toc144730186)

[3.2.9 Inference details 10](#_Toc144730187)

[3.3 Low complexity operation point neural network-based loop filter set 10](#_Toc144730188)

[3.3.1 Neural network 10](#_Toc144730189)

[3.3.2 Residue scaling 11](#_Toc144730190)

[3.3.3 Combination with deblocking filters 11](#_Toc144730191)

[3.3.4 Inference details 12](#_Toc144730192)

[3.4 Neural network-based intra prediction 12](#_Toc144730193)

[3.4.1 Neural network inference 12](#_Toc144730194)

[3.4.2 Preprocessing and postprocessing 13](#_Toc144730195)

[3.4.3 Adaptation of the derivation of the list of MPMs 14](#_Toc144730196)

[3.4.4 Signaling of the neural network-based intra prediction mode 15](#_Toc144730197)

[3.4.5 Transformation of the context and the neural network prediction 16](#_Toc144730198)

[3.5 Small ad-hoc deep learning (SADL) library 17](#_Toc144730199)

[3.6 Content-adaptive neural network post-filter 17](#_Toc144730200)

[3.6.1 Neural network 17](#_Toc144730201)

[3.6.2 Content adaptation 18](#_Toc144730202)

[3.6.3 Inference details 18](#_Toc144730203)

[3.7 Neural network-based super resolution 19](#_Toc144730204)

[3.7.1 Neural network for luma component 19](#_Toc144730205)

[3.7.2 Neural network for chroma component 19](#_Toc144730206)

[3.7.3 GOP level encoding resolution decision 19](#_Toc144730207)

[3.7.4 Inference details 20](#_Toc144730208)

[3.8 High Operating Point model 21](#_Toc144730209)

[3.8.1 Overview 21](#_Toc144730210)

[3.8.2 Architecture 21](#_Toc144730211)

[3.8.3 Model usage aspects 22](#_Toc144730212)

[3.9 Inference details 23](#_Toc144730213)

[4 Training description of Neural Network-based Video Coding Software 24](#_Toc144730214)

[4.1 Neural network-based loop filter set 0 24](#_Toc144730215)

[4.2 Neural network-based loop filter set 1 24](#_Toc144730216)

[4.2.1 Regular filters 24](#_Toc144730217)

[4.2.2 Temporal filter 25](#_Toc144730218)

[4.3 Low operating point neural network-based loop filter set 25](#_Toc144730219)

[4.4 Neural network-based intra prediction 26](#_Toc144730220)

[4.5 Content-adaptive neural network post-filter 27](#_Toc144730221)

[4.6 Neural network-based super resolution 27](#_Toc144730222)

[4.7 High Operating Point model training 27](#_Toc144730223)

[4.7.1 Overview 28](#_Toc144730224)

[4.7.2 Training characteristics 28](#_Toc144730225)

[4.7.3 Prerequisite 29](#_Toc144730226)

[4.7.4 Training Model stage I 30](#_Toc144730227)

[4.7.5 Training Model stage II 31](#_Toc144730228)

[4.7.6 Training stage III 32](#_Toc144730229)

[5 References 32](#_Toc144730230)

# 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).

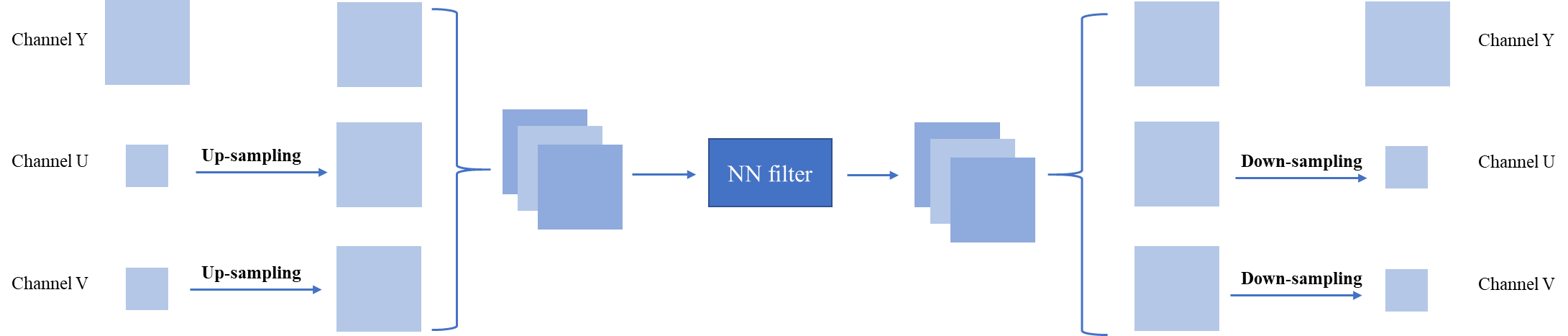
This document provides an algorithm description, an encoder-side description, and a training description of the NNVC, which serves as a tutorial for the algorithm and encoding model implemented in the NNVC software, as well as the training method of the tools in NNVC software. The purpose of this document 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.

# Algorithm description of Neural Network-based Video Coding Software

## 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.



*Figure. 1 the pre-processing and post-processing units*

### Neural network

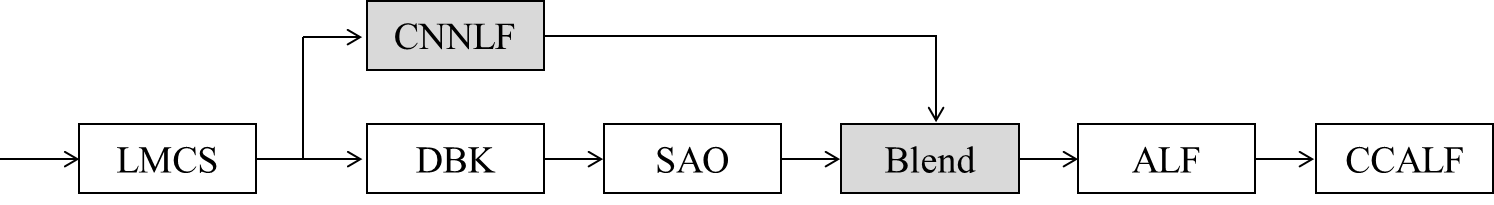
The network structure of the CNN filter is shown in Figure. 2. Along with the reconstructed image (rec\_yuv), additional side information is also fed into the network, such as the prediction image (pred\_yuv), slice QP, base QP and slice type. In the ResBlock, the number of channels firstly goes up before 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



*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

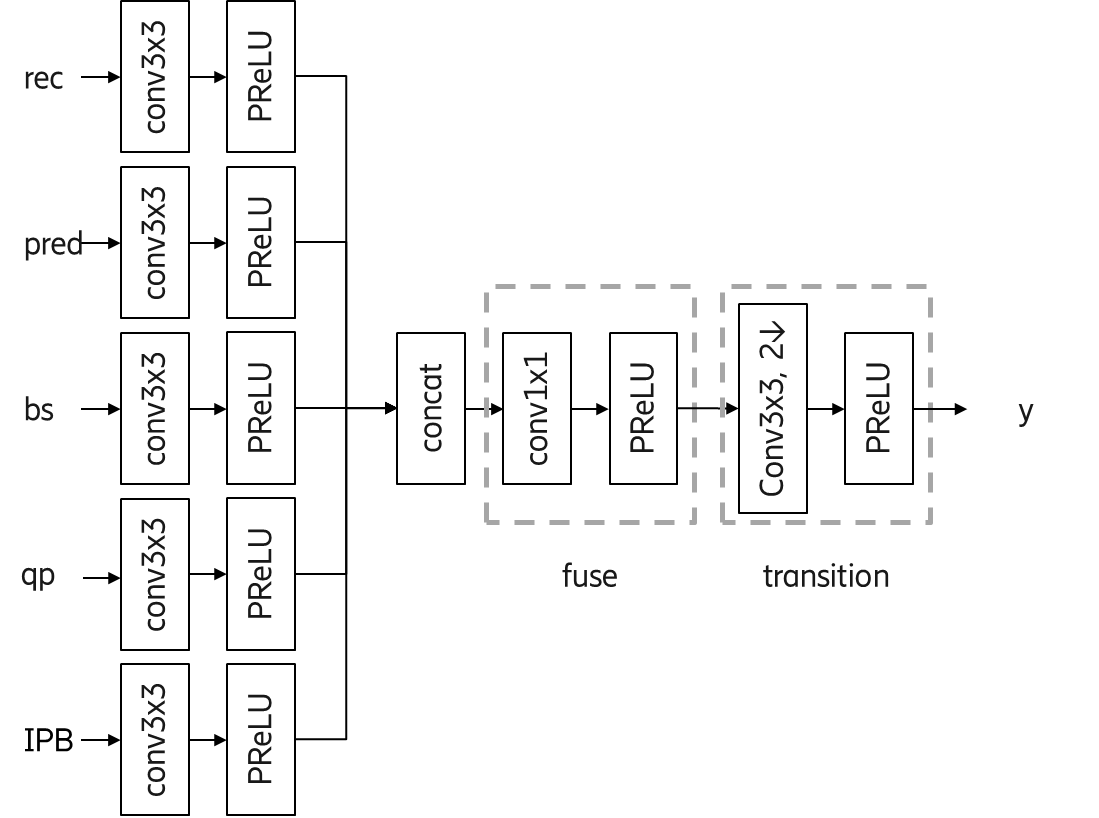
|  |  |  |
| --- | --- | --- |
| **Network Information in Inference Stage** | | |
| Mandatory | HW environment: | |
| GPU Type | N/A |
| Framework: | SADL |
| Number of GPUs per Task | 0 |
|  |  |
| Number of Parameters (Each Model) | 1.9M |
| Total Parameter Number | 1.9M, one model in total |
| Parameter 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

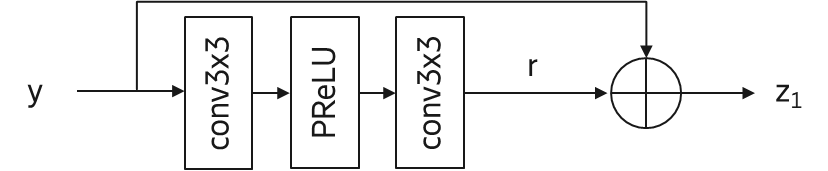
### Neural network for luma component

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

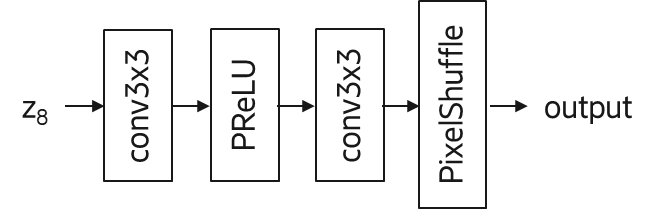
The inputs of the luma network comprise the reconstructed luma samples (rec), the prediction luma samples (pred), boundary strengths (bs), QP, and the block type (IPB). The numbers of feature maps and residual blocks are set as 96 and 8 respectively. The structure of the luma network is depicted in Figure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



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



*(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.*



*(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: |  |

## Low complexity operation point neural network-based loop filter set

### Neural network

The network structure of the low complexity operation point CNN based loop filter is shown in Figure 7.. The inputs to the loop filter are reconstructed luma and chroma samples (rec\_yuv), boundary strength information for luma and chroma (3 planes) and slice QP plane. Since the resolutions of luma and chroma for YUV420 format are different, the reconstructed luma samples are decomposed into four smaller planes to match the resolution of chroma plane before filtering.

The network consists of a 3x3 CNN input layer which takes in 10 input layers with M(72) output features. This is followed by n(11) hidden layers, each hidden layer consists of 1x1 pointwise convolution with wide activation (M=72), a second 1x1 pointwise convolution with reduced output feature map (K=24) and 3x3 convolution layers which are decomposed and fused into separable layers as follows.

The 3x3 convolutions of each hidden layer are decomposed into 4 layers with rank *R* followed by fusion of adjacent 1x1 convolution as shown in :

* 1st layer: 1x1x*K*x*R* pointwise convolution
* 2nd layer: 3x1x*R*x*R* separable convolution
* 3rd layer: 1x3x*R*x*R* separable convolution
* 4th layer: 1x1x*R*x*K* pointwise convolution

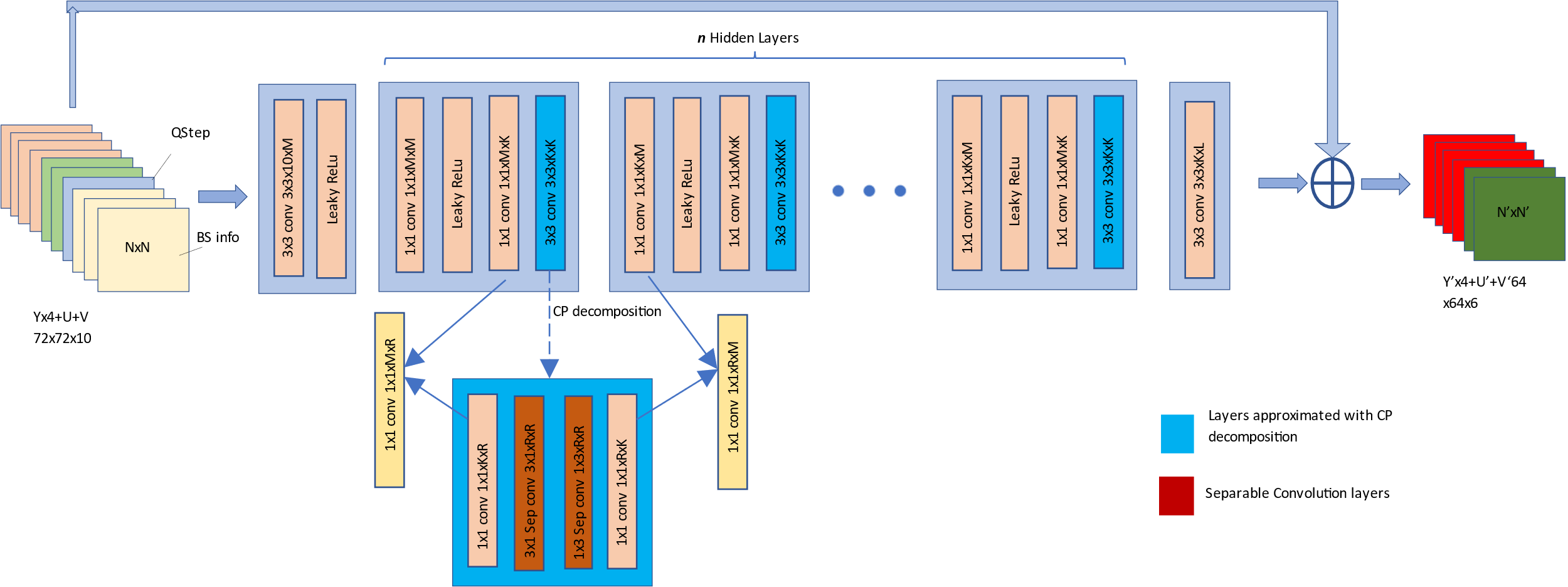


Figure .Architecture of low complexity CNN filter set (CP Decomposition + fusion of 1x1 conv layers)

The final output layer consists of 3x3 convolution layers which outputs filtered samples for L=6 planes (4 luma and 2 chroma planes) used for final residual scaling.

### 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 8**Error! Reference source not found.**, 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 8 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 | 105132 |
| Parameter Precision (Bits) | float: 32  int: 16 |
| Memory Parameter (MB) | float: 0.401 MB  int: 0.201 MB |
| Multiply Accumulate (kMAC/pixel) | 16.2 |
| Optional |  |  |
| Total Conv. Layers | 46 |
| Total FC Layers | 0 |
| Total Memory (MB) |  |
| Batch size: | 1 |
| Patch size | 72x72 (4 luma and 2 chroma planes) |
| Changes to network configuration or weights required to generate rate points |  |
| Peak Memory Usage |  |
| Other information: |  |

## 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 3: 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 4: 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 5: 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 chroma CB whose top-left pixel is at position in the current chroma channel, the intra prediction mode signaling in chroma is split into two cases.

* If the luma CB collocated with this chroma CB 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 .

## 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 | ~6700 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 |
| 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 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.

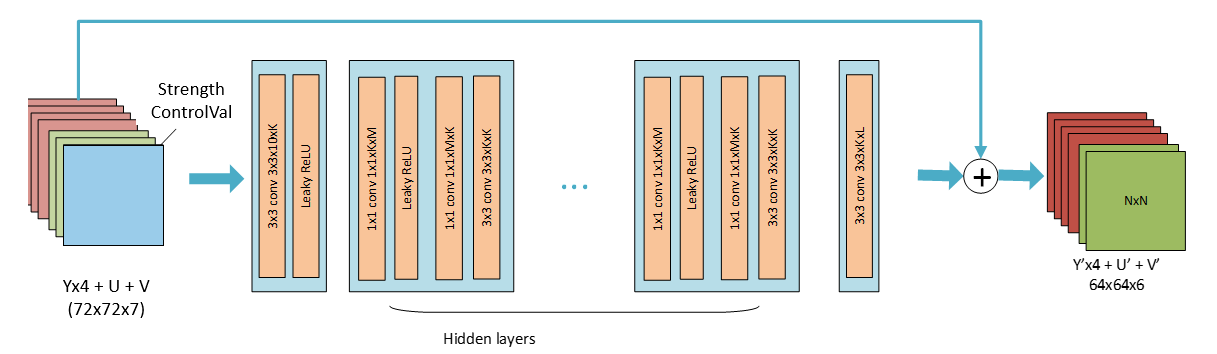


Figure . Architecture of the content-adaptive post-filter

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

There are two networks for NN-based super resolution, one for luma component and one for chroma component.

The designed network structure for luma is shown in Figure 1. Along with the low-resolution reconstruction , the low-resolution prediction , slice QP and base QP are also fed into the proposed network. The RPR reconstruction upsampled by conventional RPR mechanism is adopted to generate the high-resolution reconstruction . The number of resblocks is set to 24, and the channels K and M in the resblock are set to 64 and 192 respectively. Two models are traind for I slices and B slices, and the base QP huatuis removed for the I slices.

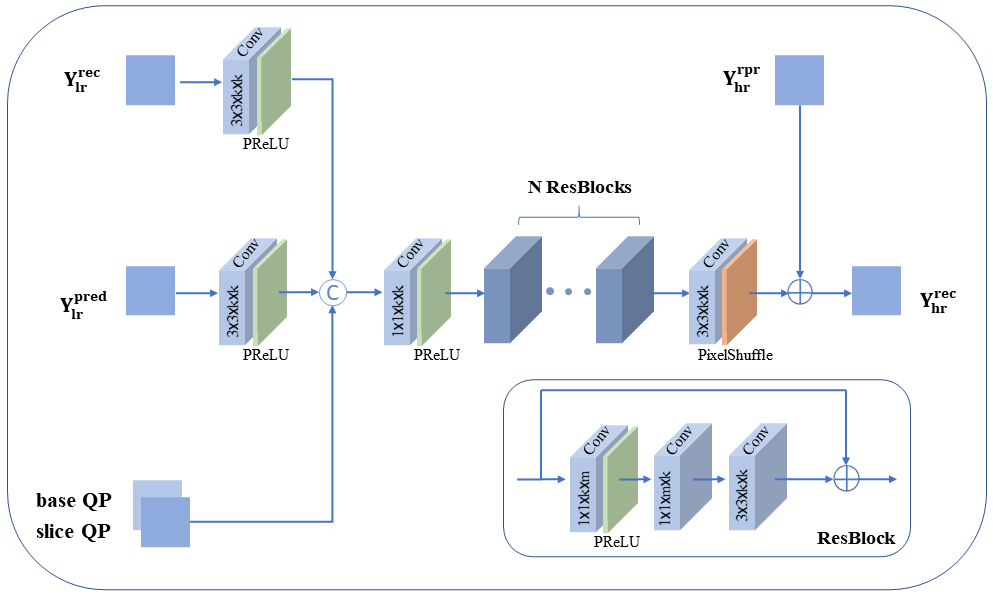


Figure 1. Architecture of the super-resolution filter

### Neural network for chroma component

The designed network structure for chroma is similar to the network structure for luma. Since the cross-component correlation is beneficial to the super resolution of chroma components, the luma reconstruction is fed into the network. Besides, the slice type is also taken as one input for better generalization.

### GOP level encoding resolution decision

One of resampling scale factors {×1.0 (original size) and ×2.0 (half size)} is selected at GOP level and the same scale factor is applied on all frames in a common one GOP. To determine the resampling factor, 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 factor decision. The scale factor decision process is shown in Figure 2.

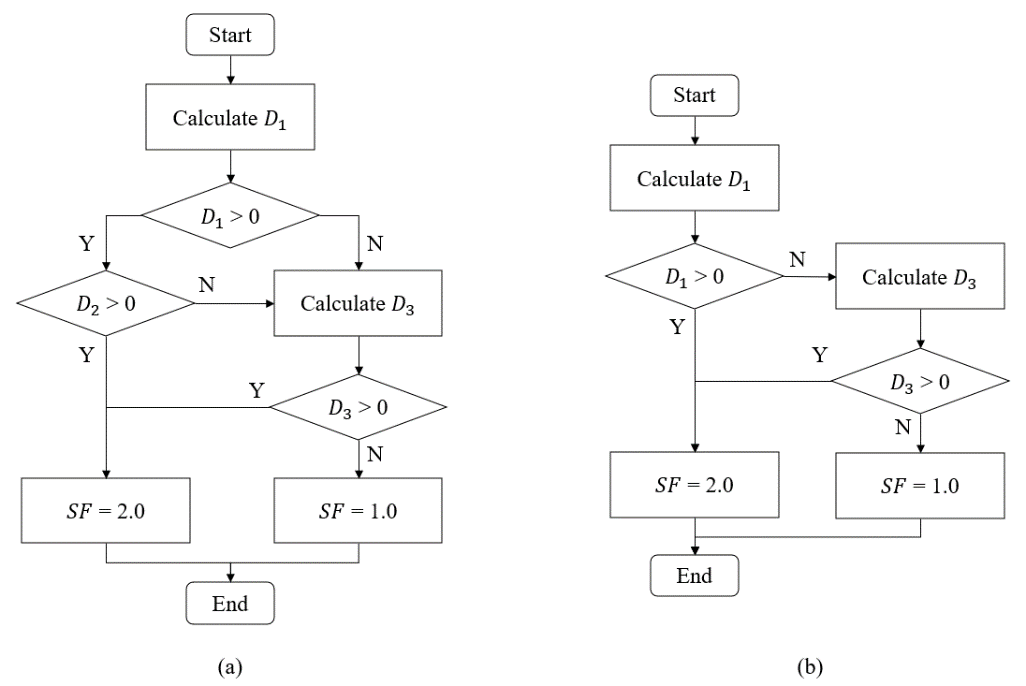


Figure 2. (a) Scale factor decision for RA. (b) Scale factor decision for AI.

Specifically, the calculation formulas used in Figure 1 are shown as follows.

where and represent down-up scaled luma and chroma PSNR for the first frame of GOP, respectively. and represent predefined threshold for luma and chroma PSNR, respectively. represents another predefined criteria for luma PSNR and represents predefined criteria for QP. is the initial QP for the first frame of GOP and in Figure 1 is the determined resampling scale factor.

### 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 1.

Table 1. Network Information of super resolution 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) | luma model for I slice: 1.49 M  luma model for B slice: 1.49M  chroma model: 1.50M |
| Total Number of Parameters (All Models) | 4.48M |
| Parameter Precision (Bits) | float: 32, int: 16 |
| Memory Parameter (MB) | float: 17.14MB, 3 models in total  int: 8.60MB, 3 models in total |
| Multiplay Accumulate (MAC)/pixel | 469 (assuming frame-level input)  593 (assuming block-level input) |
| Optional |  |  |
| Total Conv. Layers | 76 |
| 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) |  |

## High Operating Point model

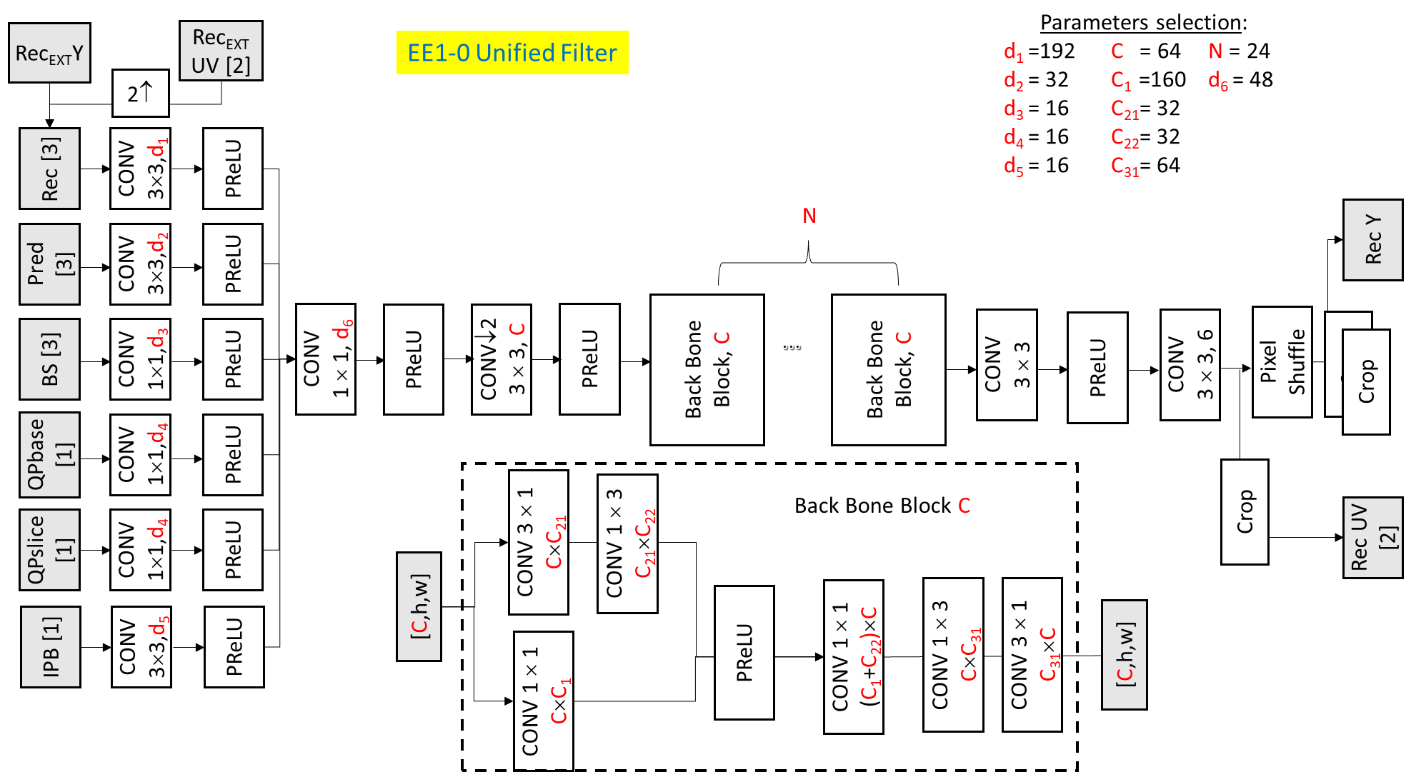
### Overview

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

### Architecture

High Operating Point model structure is given by the figure below.



The Table 4 gives the characteristics of the model.

Table 4 **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 |

### 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

## 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 | 477 kMAC/pix (assuming block-level input) |
| Optional |  |  |
| Total Conv. Layers | 154 |
| Total FC Layers | 0 |
| Batch size: | 1 |
| Patch size | 128128, 256256 (+8 border) |

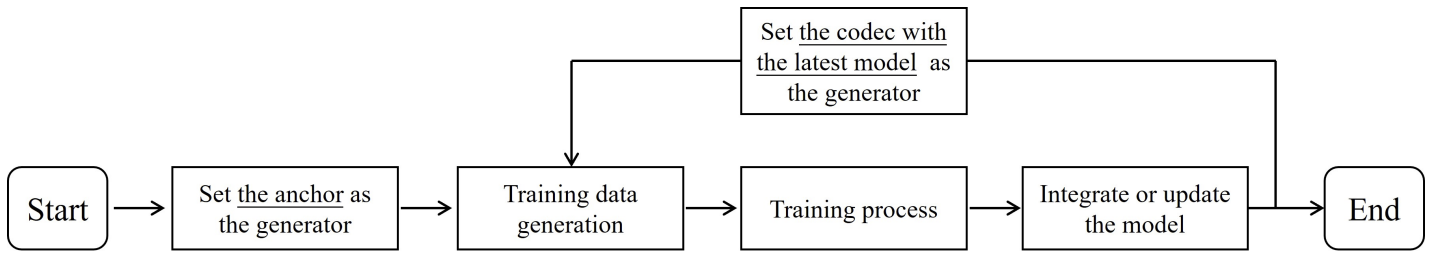
# Training description of Neural Network-based Video Coding Software

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.

## 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.



*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.

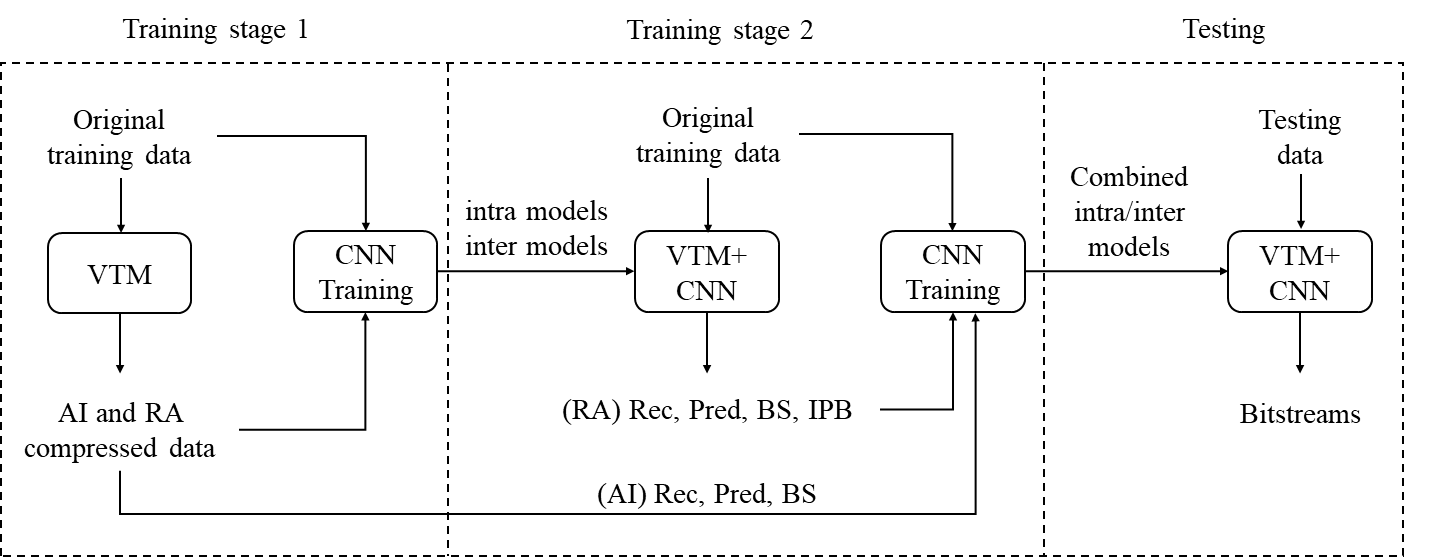
In the iterative training stage, NNVC (--NnlfOption=1) with the integrated NN model from previous training stage is used to compress all training images under random-access configuration and the training data for I slices is still used from initial training stage. Theoretically, the more times of the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

## 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.



*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.

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

The models are trained iteratively for better performance. Joint models are used for luma and chroma. 2 models are used for Intra(AI) and 2 models are used for Inter(RA). Two models for each configuration is trained by splitting the training data based on the configured QP. Hence out of two models, one is trained for lower QP points and the other for higher QP points. The intra models are trained with DIV2K dataset and inter models are trained with BVI-DVC dataset.

The overview of intra models training is shown in Figure 7. The raw DIV2K image data was compressed using AI configurations with NNVC(--NnlfOption=0) to create the training examples for AI. The baseline model with 3x3 convolution layers before decomposition is trained initially. The 3x3 convolutions of each hidden layer are decomposed into 4 layers(CP decomposition) using PARAFAC followed by fusion of adjacent 1x1 convolution followed by training to obtain intra models.

The overview of inter models training is shown in Figure 8. The raw BVIDVC video data was compressed using RA configurations with NNVC(--NnlfOption=3) to create the training examples for RA. 1st stage RA training dataset is generated with trained intra models. 2nd stage RA training dataset is generated using trained intra models and inter models from 1st stage RA training. The initial weights for the 1st stage RA is got from trained intra models. The initial weights for the 2nd stage RA is got from the 1st stage trained RA models.

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 7 Overview of training Intra(AI) models

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 8 Overwiew of training Inter(RA) models

## 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.

## Neural network-based super resolution

The training of neural network-based super resolution is straightforward. The NNVC software with RPR enabled is used to generate training datasets. Two luma models are trained for I slices and B slices respectively and only one model is trained for chroma component.

## High Operating Point model training

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 10 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.

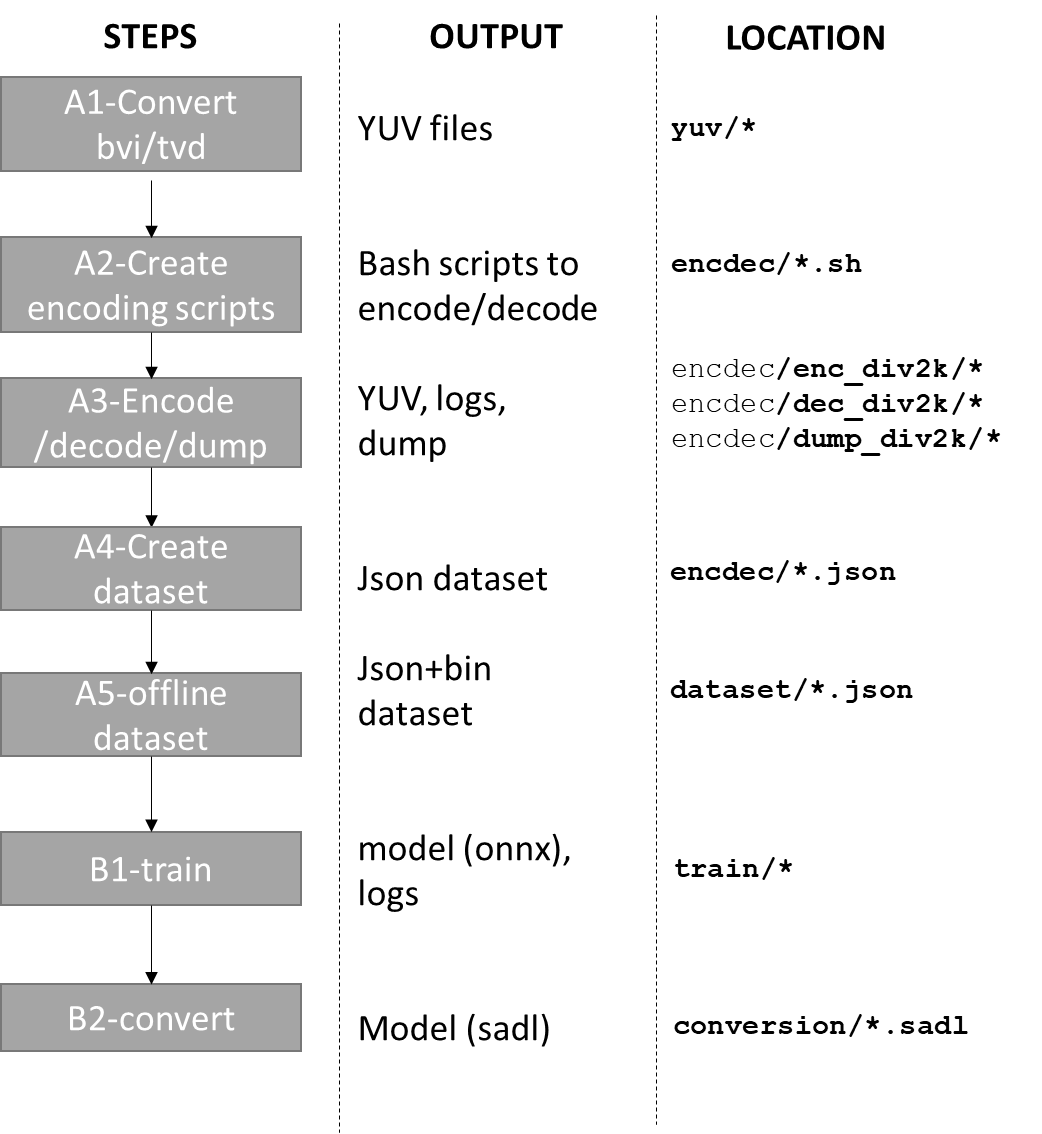
### Training Model stage I

A screenshot of a computer program

Description automatically generated

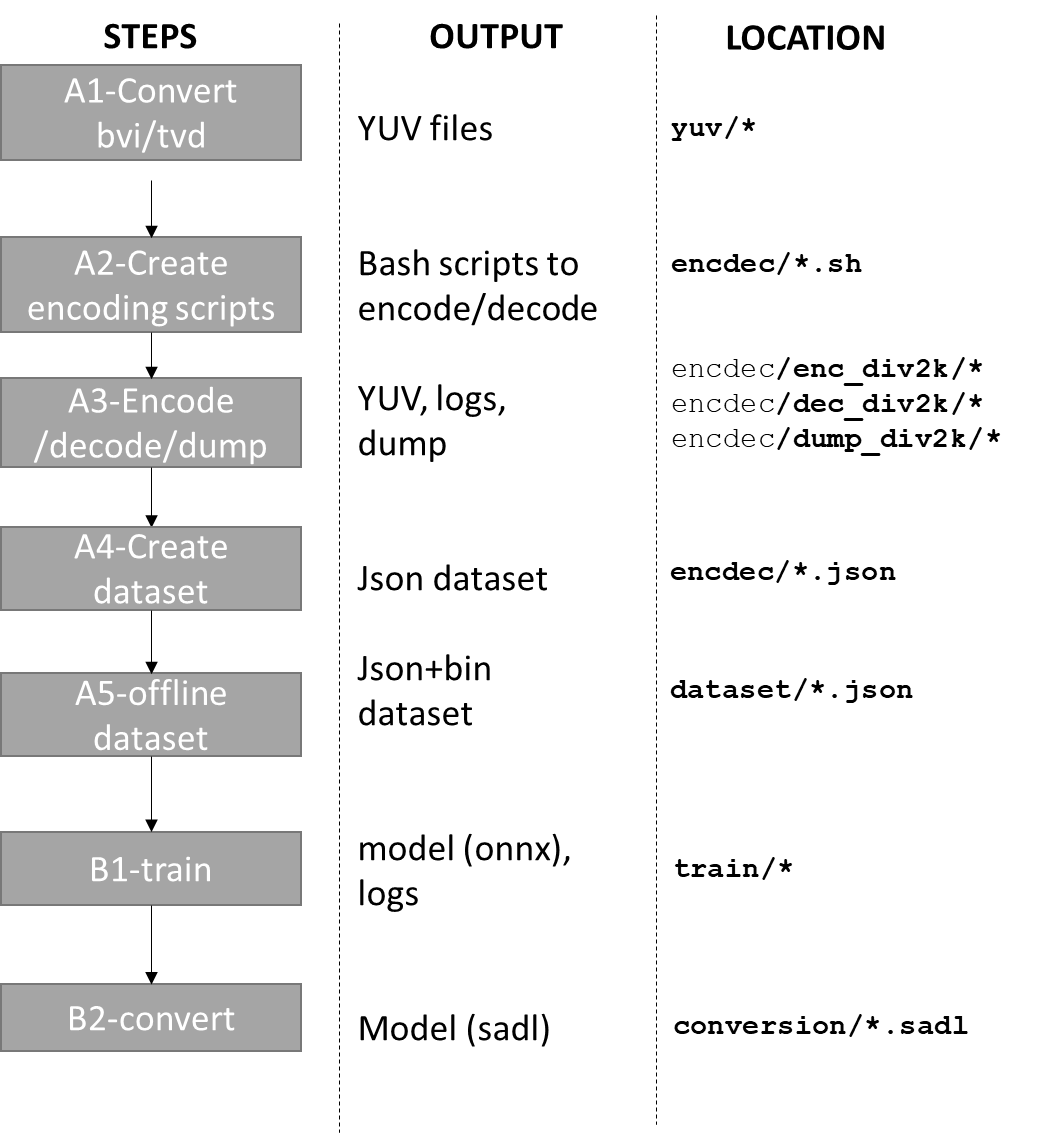
* 1. Data extraction for intra from vanilla VTM
     1. encoder turns on md5sum, just NNVC-5.0 all NNVC tools disabled, encoder\_intra\_vtm.cfg, just first frame
     2. QPs = 22,27,32,37,42 + 19,24,29,34,39
     3. DIV2K (4:4:4 RGB 🡪 YUV420 10 bits, version ffmpeg to be specified,
     4. Rotation, 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. Learning Rate scheduler: 0.1 decay at epoch 30 and 34
     4. Norm switch: L1 up to epoch 37, then L2
     5. 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. Learning Rate scheduler: 0.1 decay at epoch 15 and 17
     5. Norm switch: L1 up to epoch 18, then L2
     6. 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. Learning Rate scheduler: 0.1 decay at epoch 15 and 17
     5. Norm switch: L1 up to epoch 18, then L2
     6. 20 epochs total

# 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