**A CNN-Based In-Loop Filter Approach for AV1 Video Codec**

Guangyao Chen, Dandan Ding

Hangzhou Normal University

Jan.16, 2019

This is a document describing our work “CNN-Based In-Loop Filter Approach for AV1 Video Codec”. The document consists of several parts which are list below. All of our code and models are available on website: <https://github.com/IVC-Projects/AV1_CNN_in-loop_filter>. If you have any question, please report bugs/questions/suggestions to [gangyaochen@gmail.com](mailto:gangyaochen@gmail.com) and cc to DandanDing@hznu.edu.cn.

# Content of this document

* Step 1: Prerequisites
* Step 2: the data set for training
* Step 3: Training settings
* Step 4: Incorporating CNN into AV1
* Step 5: Test
* Step 6: Usage of Demo

# Step 1: Prerequisites

* Windows or Linux system. All following operations will be done on a Linux system unless otherwise stated.
* Download the AOM source from <https://aomedia.googlesource.com/aom>

Our version is version 15-July 2018.

* Python

Our Python version is: Python 3.6.6.

* TensorFlow

Our tensorflow version is '1.6.0'.

* PIL

# Step 2: The data set for training

We build several databases for training. We collect 300 video sequences and the first 200 frames of each sequence are encoded in AV1 reference software under QP = {32, 37, 47, 52}, respectively, with in-loop filter off. Afterwards, around 40000 raw reconstructed frames are generated for intra/inter coding type at each QP. We select one frame per 20 frames in each sequence and form a dataset including 2000 pictures. For the intra coding, we combine the DIV2K dataset and the above 2000 pictures together for training. In terms of the inter coding, the 2000 frames are used for training.

Notice that comparable performance can be achieved if only DIV2K data set is used in intra coding.

In our code, the network directly reads samples from the pictures in the database. Of course, you can employ TFrecords instead. In our implementation, we randomly take 64 patches which is 35x35 size in each picture. Assume that the number of pictures is W, thus 64 x W patches will be handled within an epoch.

Similarly, the original images which are not compressed also go through the above steps to generate labels.

# Step 3: Training settings

Our overall CNN structure is similar to VDSR , except that the network depth is adapted to QP levels. In our implementation, the depth is set to {15, 20, 25, 25} for QP = {32, 37, 47, 52}, respectively. Besides, different models are trained for intra and inter coding. Taking QP 52 which involves 25 layers for example, we give a detailed description on each layer, as tabulated in .

Table I Description of our network structure using 25 convolutional layers at QP=52

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Layer No. | Operation | Feature maps | Kernel size | Stride | Weight  initialization | Bias initialization |
| 1 | Convolution | 64 | 3x3 | 1 | MSRA, | constant, 0 |
| ReLU |  |  |  |  |  |
| 2-24 | Convolution | 64 | 3x3 | 1 | MSRA, | constant, 0 |
| ReLU |  |  |  |  |  |
| 25 | convolution | 1 | 3x3 | 1 | MSRA, | constant, 0 |

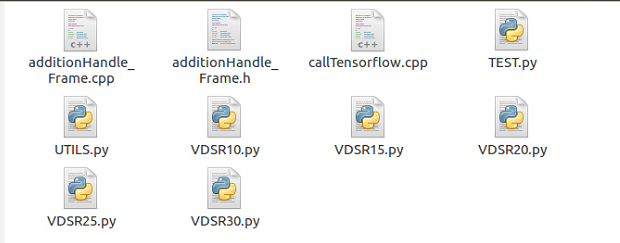
To train the model, the initial learning rate is set to 0.0001. The learning rate is adjusted with the step strategy using gamma=0.5. In our implementation, the learning rate is multiplied by 0.5 every 180 epochs in QP=52 for the intra coding. In terms of the inter coding, the learning rate is halved per 80 epochs in QP=52. And, Small QP may converge faster.

Besides, the adaptive moment estimation (Adam) method is employed for back propagation, and the momentum is set to 0.9. Gradient clipping is optional in the training, which is even not used in our implementation.

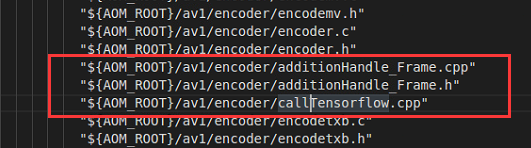
# Step 4: Incorporating CNN into AV1

Below we will present how to incorporate CNN model into AV1 codec. The essential idea is using Python/C API in AV1 to call the Python script. After the anchor in-loop filter, the reconstructed samples in buffer are sent to CNN model for enhancement and then sent back to the buffer. More specifically,

1. Add additionHandle\_Frame.h, additionHandle\_Frame.cpp, callTensorflow.cpp and CNN related code files.



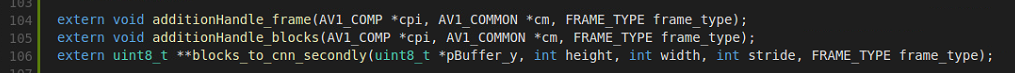
1. Put the above three files and corresponding CNN files into the ${AOM ROOT}/ av1/encoder directory.
2. Modify ${AOM ROOT}/av1/av1.cmake by adding the downloaded three file names to list (APPEND AOM\_AV1\_ENCODER\_SOURCES ).

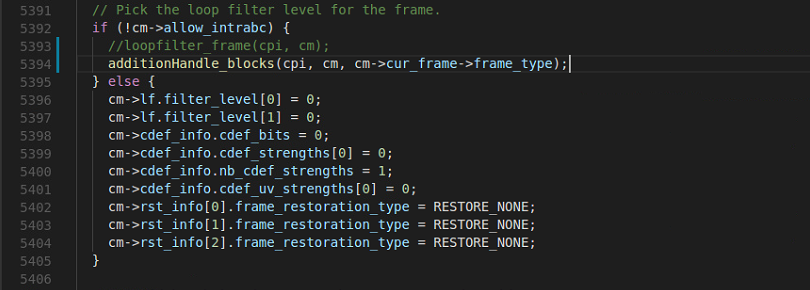


1. Add the python dependent library at the end of CmakeLists.txt.



1. Use Cmake to build a project.
2. Modify encoder.c, comment out the entry of loop filter and replace it using your own function.





1. Conduct “make”.

# Step 5: Test

* Intra coding

We apply the CNN model to every single frame in intra coding.

* Inter coding

In inter coding, the output of an in-loop filter, also referred to as the restored frame, is used as reference for coding subsequent frames, which will affect the encoding performance of subsequent frames. We propose to selectively enhance some frames.

Consequently, we select frame 2, 6, and 10 for enhancement if the Golden Frame Group is 12 (GOP size=12). For the rest frames, the baseline in-loop filter will be applied. If the GOP size is 8, only frame 2 and frame 6 will be enhanced.

It is worth noting that in our AV1 version, the PSNR of the first key frame in inter coding is quite high, which seems a bug in QP setting. We resort to models of small QP to enhance the first frame, which attains advanced results.

Finally, the GPU's memory may be insufficient when testing an image in high resolution. To address such issue, we divide a large image into several small rectangles and feed them into CNN in sequence.

Here is the test commands we use (e.g., BasketballDrill\_832x480\_50.yuv sequence).

* Intra:

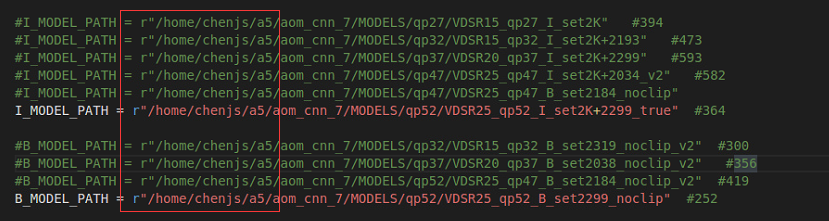
./aomenc --width=832 --height=480 --fps=50/1 --codec=av1 --test-decode=warn --cpu-used=1 --threads=0 --profile=0 --passes=2 --kf-max-dist=1 --kf-min-dist=1 --end-usage=q –psnr –v --cq-level=52 --limit=20 -o ../webm/qp52/BasketballDrill\_832x480\_I.webm ../test\_sequence/BasketballDrill\_832x480\_50.yuv >../log/qp52/BasketballDrill\_832x480\_I.log 2>&1

* Inter:

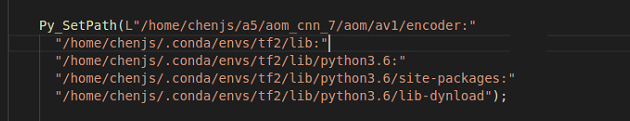
./aomenc --width=832 --height=480 --fps=50/1 --codec=av1 --test-decode=warn --cpu-used=1 --threads=0 --profile=0 --passes=2 --disable-kf --end-usage=q --psnr -v --cq-level=52 --limit=21 -o ../webm/qp52/BasketballDrill\_832x480\_I.webm ../test\_sequence/BasketballDrill\_832x480\_50.yuv >../log/qp52/BasketballDrill\_832x480\_I.log 2>&1

# Step 6: Usage of Demo

1. Download the entire folder from our website: https://github.com/IVC-Projects/AV1\_CNN\_in-loop\_filter.
2. There are some places you might need to modify.
   1. Modify the path of the models in path-to-project/aom/av1/encoder/TEST.py. We suggest using the absolute path.



* 1. Modify the path of the dependent libraries in path-to-project/aom/av1/encoder/encoder.c.



1. Go to path-to-project/aom\_build directory and run make command.
2. You can add test sequences to the path-to-project/sequences directory and test with the commands described above.

# References

1. Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee, “Accurate image super-resolution using very deep convolutional networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp.1646-1654.