Deep Learning Based HM Encoder (Intra-mode)

This encoder is used for evaluating the performance of our deep ETH-CNN based approach [1] (improved from the conference version [2] published on IEEE ICME) at All-Intra configuration. The main part is modified from the standard reference software HM 16.5 [3], coded with C++. The proposed ETH-CNN is realized based on Tensorflow, coded with Python 3.5. For evaluating the performance of our deep learning based approach, the Python program is invoked inside the HM encoder. To encode a YUV file, the probability of CU partition for all the frames is predicted in advance, before the encoding process in HM really starts. Compared with the upper and lower thresholds at three levels, the probability is read to finally determine the CU partition by HM. In this way, most redundant checking of RD cost checking can be skipped, thus save the overall encoding time significantly.

Process

1. Before encoding the first frame, HM invokes the Python program *video_to_cu_depth.py* via a command line with some essential paramaters, containing YUV file name, frame width, frame height and QP.
2. The Python program reads the YUV file and other essential parameters, to predict the probability of CU partition for all the frames and save it in file *cu_depth.dat*.
3. HM encodes all the frames according to the predicted CU partition probability from *cu_depth.dat*, thus simplifying the RDO process by skipping redundant checking of RD cost.

Source

In HM 16.5, four C++ files have been modified, as below.

- source\App\TAppEncoder\TAppEncCfg.cpp
- source\Lib\TLibCommon\TComPic.h
- source\Lib\TLibEncoder\TEncGOP.cpp
- source\Lib\TLibEncoder\TEncCu.cpp

Among them, *TAppEncCfg.cpp* directly invokes the Python program.

Note: the codes are based on the assumption that the default Python is Python 3 in the system (Windows or Linux). If it is false, please modify the 2319th line of *TAppEncCfg.cpp* by changing the string *python* into *python 3*, and rebuild the HM.

Also, two Python files are included for predicting the CU partition with the proposed deep networks. The top file is *video_to_cu_depth.py*, monitoring the overall procedure of adopting ETH-CNN, together with necessary steps such as file reading/writting, data transferring, network feeding, etc. The specific network architecture is defined in *net_CNN.py*.

For more details, please refer to the comments in these source codes.

This program is used to evaluate the performance of our deep ETH-CNN based approach at All-Intra configuration.

Running Instructions

1. Install Tensorflow. Versions $\geq$ 1.4.0 are recommanded.

2. Path into *HM-16.5_AI\bin\vc10\x64\Release*

3. Set upper/lower thresholds for 3-level CU partition in file *Thr_info.txt*

   Format: [8 $\bar{\alpha}_1$ α_1 $\bar{\alpha}_2$ α_2 $\bar{\alpha}_3$ α_3]

   Example: [8 0.5 0.5 0.5 0.5 0.5 0.5]

4. Run *TAppEncoder.exe* on Windows or *TAppEncoderStatic* on Linux.

   Examples: *RUN_LDP.bat* and *RUN_LDP.sh*

Deep Learning Based HM Encoder (Inter-mode)

This encoder is used for evaluating the performance of our deep ETH-CNN + ETH-LSTM based approach [1] at Low-Delay-P configuration. The main part is modified from the standard reference software HM 16.5, coded with C++. The proposed ETH-CNN and ETH-LSTM are realized based on Tensorflow, coded with Python 3.5. For evaluating the performance of our deep learning based approach, the HM and the Python program are linked together via sharing intermediate information when running both the programs. When encoding a YUV file, the CU partition is predicted frame-wise in accord with the encoding process in HM. For a certain frame, a simplified setting is adopted for quick pre-encoding, to obtaining the residue of this frame in HM. Next, the residue is fed into ETH-CNN + ETH-LSTM in the Python program. Then the Python program predicts the probability of CU partition for this frame. Compared with the upper and lower thresholds at three levels, the probability is read to finally determine the CU partition by HM. In this way, most redundant checking of RD cost can be skipped, thus save the overall encoding time significantly.

Process

1. During encoding one frame, HM adopts a simplified setting (CU size and PU size are all the maximum, 64 × 64, except on the right/bottom edge without full-size CTUs) to quickly pre-encode the frame, for obtaining the residue. Then the residue frame is saved into file *resi.yuv*. Note that the quick pre-encoding for residue is not included in the standard HM. Instead, it is introduced by our approach, to provide the input for ETH-CNN at inter-mode.

2. In HM, some essential parameters are wirtten into file *command.dat*.

   Format: Frame_index Frame_width Frame_height QP [end]

   Example: 19 416 240 22 [end]

3. HM generates a signal file *pred_start.sig*, indicating the residue file and command file are ready and the Python program can start to run. Here the Python program need only check whether the siginal file exists rather than the content. So the siginal file can be null, for simplicity.

4. Once *pred_start.sig* is detected by the Python program, some essential files are read, containing the residue *resi.yuv*, the command *command.dat* and the previous state of ETH-LSTM *state.dat* (if have).

5. The residue frame and the LSTM state are fed into ETH-CNN + ETH-LSTM. Then ETH-LSTM runs ahead for one time step, and the file *state.dat* is updated. Also, the predicted CU partition probability for the whole frame, is saved in file *cu_depth.dat*.

6. After updating the state and generating *cu_depth.dat*, the Python program generates an end signal *pred_end.sig*, indicating the prediction for current frame is finished.

7. Once *pred_end.sig* is detected by HM, the probability of all possible CUs are read from *cu_depth.dat*, based on which the CU partition is determined.

8. HM encodes the current frame according to the predicted CU partition, thus simplifying the RDO process by skipping redundant checking of RD cost.

Source

In HM 16.5, five C++ files have been modified, as below.

- source\Lib\TLibCommon\TComPic.h
- source\Lib\TLibEncoder\TEncGOP.cpp
- source\Lib\TLibEncoder\TEncCu.cpp
- source\Lib\TLibEncoder\TEncSearch.cpp
- source\Lib\TLibEncoder\TEncSlice.cpp

Among them, the *TEncGOP.cpp* directly invokes ETH-CNN + ETH_LSTM.

Also, three Python files are included for predicting the CU partition with the proposed deep networks. The top file is *resi_to_cu_depth_LDP.py*, monitoring the overall procedure of adopting ETH-CNN + ETH-LSTM, together with necessary steps such as file reading/writting, data transferring, network feeding, etc. The specific network architecture is defined in *net_CNN_LSTM_one_step.py*, and some constants are stored in *config.py*.

For more details, please refer to the comments in these source codes.

This program is used to evaluate the performance of our deep ETH-CNN based approach at All-Intra configuration.

Running Instructions

1. Install Tensorflow. Versions $\geq$ 1.4.0 are recommanded.
2. Path into *HM-16.5_LDP\bin\vc10\x64\Release*
3. Set upper/lower thresholds for 3-level CU partition in file *Thr_info.txt*

   Format: [108 $\bar{\alpha}_1$ α_1 $\bar{\alpha}_2$ α_2 $\bar{\alpha}_3$ α_3]

   Example: [108 0.4 0.6 0.3 0.7 0.2 0.8]
4. Ensure there exists no any signal file, *pred_start.sig* or *pred_end.sig*, for avoiding unproper behavier of communication between the HM encoder and the python program *resi_to_cu_depth_LDP.py*.
5. Run *resi_to_cu_depth_LDP.py* with Python 3.5, and *Initializing. Please wait...* is shown.
6. Wait for about 1~10s, until *Python: Tensorflow initialized.* is shown.
7. Run *TAppEncoder.exe* on Windows or *TAppEncoderStatic* on Linux.

Examples: *RUN_LDP.bat* and *RUN_LDP.sh*

References

[1] M. Xu, T. Li, Z. Wang, X. Deng, R. Yang and Z. Guan, (2018). Reducing Complexity of HEVC: A Deep Learning Approach. arXiv preprint arXiv: 1710.01218.

[2] T. Li, M. Xu and X. Deng, "A deep convolutional neural network approach for complexity reduction on intra-mode HEVC," 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, Hong Kong, 2017, pp. 1255-1260.

[3] JCT-VC, “HM Software,” [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.5/, 2014, [Accessed 5-Nov.-2016].