Research and examination on implementation of super-resolution models using deep learning with INT8 precision

Shota HIROSE, Naoki WADA, Jiro KATTO
School of Fundamental Science and Engineering Faculty of Science and Engineering, Waseda University
Shinjuku, Tokyo, Japan
syouta.hrs@akane.waseda.jp

Heming SUN
Waseda Research Institute for Science and Engineering
Waseda University, Tokyo, Japan
JST, PRESTO, Saitama, Japan
hemingsun@aoni.waseda.jp

***Abstract*— Fixed-point arithmetic is a technique for treating weights and intermediate values as integers in deep learning. Since deep learning models generally store each weight as a 32-bit floating-point value, storing by 8-bit integers can reduce the size of the model. In addition, memory usage can be reduced, and inference can be much faster by hardware acceleration when special hardware for int8 inference is provided. On the other hand, when inferences are carried out by fixed-point weights, accuracy of the model is reduced due to loss of dynamic range of the weights and intermediate layer values. For this reason, inference frameworks such as TensorRT and TensorFlow Lite, provide a function called "calibration" to suppress the deterioration of the accuracy caused by quantization by measuring the distribution of input data and numerical values in the intermediate layer when quantization is performed. In this paper, after quantizing a pre-trained model that performs super-resolution, speed and accuracy are measured using TensorRT. As a result, the trade-off between the runtime and the accuracy is confirmed. The effect of calibration is also confirmed.**

Keywords— Tensor RT, Quantization, Super resolution, Real-time inference

I. INTRODUCTION

Image super-resolution is an ill-posed problem for reconstructing the original image from downscaled image. To achieve this, it is needed to add the lost information(mainly high frequency signal). SRCNN[1] is the first successful approach in performing super-resolution using convolutional neural network(CNN). SRCNN itself enhances algorithmically upsampled images. On the other hand, ESPCN[2] uses PixelShuffle (called DepthToSpace in TensorFlow) for downsampled image inputs. ESPCN is much faster than SRCNN because the computational cost of ordinary CNN is proportional to the resolution of the input image. In addition, directly using downsampled images is better in Peak Signal-to-Noise Ratio (PSNR) because refining upsampled images needs larger size of convolution kernel. Therefore, ESPCN is better than SRCNN in throughput and PSNR. After the invention of ESPCN, super-resolution models became deeper and deeper for better PSNR. For example, RCAN[3] is a very deep model for achieving very good PSNR. It has 400 convolutions in total and residual structure for extracting very detailed features of images. It has the significant accuracy but results in longer runtime due to the model structure.

Recently, super-resolution becomes convenient for enhancing images taken by mobile devices, which does not have powerful processor to run deep models. Therefore, the trade-off between runtime and PSNR is very important. This trend leads to Mobile AI 2021[4], which is a new competition style workshop as sub-area of NTIRE[5], where objective scores are calculated with consideration of runtime and accuracy. Memory usage is also a problem in using deeper model. To solve the problem, quantization is helpful. In quantization, the precision of the parameters of the model is reduced. In most cases, 8bit fixed-point(INT8) value is used in quantization because many devices can accelerate the calculation by using parallel INT8 accelerator. Because pretrained models in PyTorch, TensorFlow and so on, usually use 32bit floating-point(FP32) values to represent parameters, INT8 quantized model becomes about 4 times smaller than FP32 model without quantizaion. However, the accuracy of the model will significantly drop due to reduction of precision because float to int conversion loses dynamic range of the parameters. To avoid this, calibration is one of the solutions, in which the dynamic range is appropriately adjusted by measuring the distribution of tensors that flow through the model and calculating proper scaling value. In some frameworks including TensorRT, this calibration is automatically done when we have a representative dataset for calibration.

II. TENSORRT AND CALIBRATION

TensorRT [6] is a framework for fast inference, provided by NVIDIA. TensorRT reduces the inference time by optimizing the kernel by automatically specifying kernels suitable for the devices that perform inferences. TensorRT can fuse layers (such as series of convolutions, batchnorm, and ReLU), and has more optimizing techniques. In addition, it can convert the precision of models from 32-bit floating-point to 16-bit floating-point. In addition, on devices with Tensor cores that support INT8, the precision of models can be converted to INT8. Using INT8 precision and tensor cores, the speed of inference can be improved more than the inferences using 32-bit and 16-bit floating-point. However, a simple conversion from FP32 to INT8 will cause a loss of model accuracy. TensorRT has functions of calibration of models for avoiding significant loss of the accuracy.

III. EXPERIMENTS

A. Method

A trained super-resolution model (modified from ESPCN[2]) is converted to a TensorRT model, and super-resolution is performed. Calling CUDA Kernel many times can be bottleneck and we decided to use ESPCN-based model since ESPCN has only three convolutions. To convert

PyTorch's pretrained model to TensorRT's model, we used torch2trt [7]. Torch2trt is a PyTorch to TensorRT converter. It uses Python API of TensorRT, and it can make the usage of TensorRT model easy because it automatically converts PyTorch's model to TensorRT's model. However, direct conversion from PyTorch to TensorRT fails for our model because the parser of torch2trt for PyTorch is not complete. Torch2trt also supports the model conversion from Open Neural Network Exchange(ONNX) model because torch2trt has the parser for ONNX. ONNX is an universal format to write the structure of neural network, so it is used when pretrained model is exported to other platform. We found that the parser of torch2trt for ONNX can convert our model. Even from ONNX model, the model can be converted and INT8 precision is supported. Therefore, we first converted our pretrained model to ONNX, and then converted ONNX model to TensorRT model. We investigate how inference time and PSNR between original images and reconstructed ones varied depending on precision of the parameters of the model. NVIDIA's Jetson Xavier NX[8] is used as the device to perform inference. It has tensor cores, and it can infer using models with INT8 precision. To prepare images to be super-resolved, we create a dataset from a video called crowd_run, distributed by xiph.org [9]. Crowd_run is a video of people running, and consists of 500 frames. We divide each frame of the video and reduce the resolution of each frame from 1920x1080 to 960x540, then process the reduced image with a super-resolution model to measure the PSNR. In this experiment, we use the ESPCN_RGB model and the ESPCN_YCbCr model. ESPCN_RGB is based on ESPCN with 96 channels in the first layer, 48 channels in the second layer, and 12 channels in the third layer. ESPCN_YCbCr extracts only the luminance channel obtained by transforming the input image from RGB to YCbCr, and super-resolves only the luminance channel. The numbers of channels are 96, 48, and 4 for the first layer, the second layer and the third layer, respectively. When the precision of the parameters of the model is converted to INT8 by TensorRT, a calibration dataset is required. In this experiment, we use the validation dataset of DIV2K[10] as the correction dataset, with each image cropped to 960x540 from the center. As a comparison, we also measure the PSNR and speed of PyTorch, which supports FP32 and FP16 inference. To calculate PSNR, we used the formula shown in Eq.1 and Eq.2. The range of the value of images is from 0 to 255, so we divide squared error by 255. It is assumed that the resolution of the picture is H*W.

$$PSNR = -10\log_{10}\Sigma_{i,j}\Sigma_{RGB}\frac{(HR[i,j][RGB]-Recon[i,j][RGB])^2}{255^2*H*W*3}$$

Eq.1 The calculation of PSNR (RGB)

$$PSNR\_Y = -10\log_{10}\Sigma_{i,j}\frac{(HR[i,j][Y]-Recon[i,j][Y])^2}{255^2*H*W}$$

Eq.2 The calculation of PSNR (Y)

B. Results

The result is shown in Table.1. In this table, "PSNR_Y" represents PSNR of the luminance channel only.

Comparing ESPCN_RGB and ESPCN_YCbCr, PSNRs of ESPCN_YCbCr and the luminance channel are higher than those of RGB channels, regardless of the precision of models. In addition, in all cases except FP32 (TensorRT) and FP16 (PyTorch), the FPS of ESPCN_YCbCr is lower than ESPCN_RGB. ESPCN_YCbCr super-resolves only its luminance channel, and theoretical computational complexity of the convolutions is lower than ESPCN_RGB. However, there is an overhead of converting the color spaces between RGB and YCbCr, that makes ESPCN_YCbCr slower.

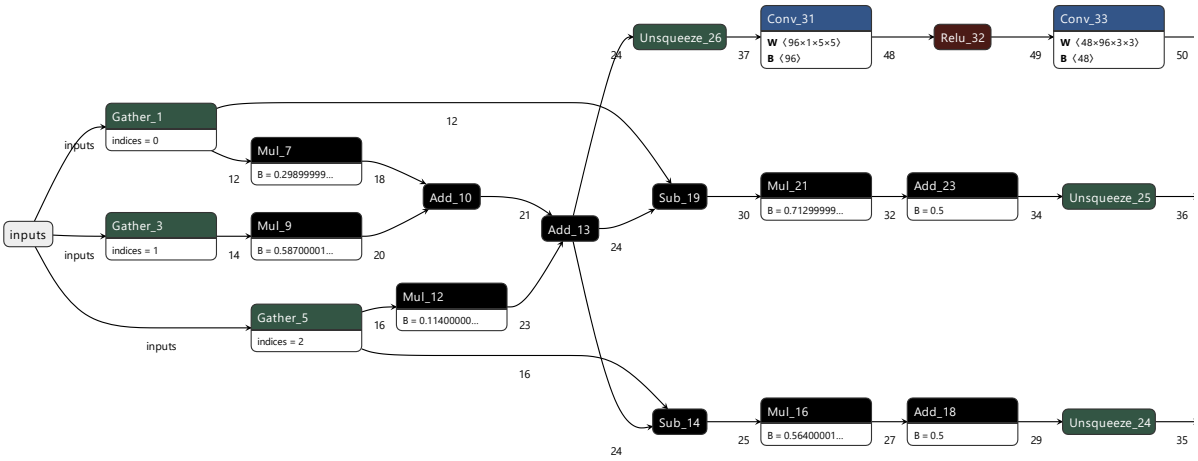
Table.1 PSNR and FPS (frames per second) among two models and bicubic upsampling

		Pytorch		TensorRT			
		FP32	FP16	FP32	FP16	INT8 (w/o calibration)	INT8 (w/ calibration)
PSNR [dB]	ESPCN_RGB	30.0797	30.0817	30.0797	30.0772	27.9382	29.7454
	ESPCN_YCbCr	30.7626	30.7586	30.7626	30.7587	29.9882	**30.5667**
	Bicubic				28.9282		
PSNR_Y [dB]	ESPCN_RGB	30.6808	30.6829	30.6808	30.6782	28.3645	30.4385
	ESPCN_YCbCr	30.7575	30.7534	30.7575	30.7536	29.9838	**30.5617**
	Bicubic				29.4030		
FPS [frames/sec]	ESPCN_RGB	10.39	16.01	14.08	28.16	51.95	**51.95**
	ESPCN_YCbCr	10.27	16.97	14.82	28.01	47.37	**47.37**
	bicubic				78.5102		

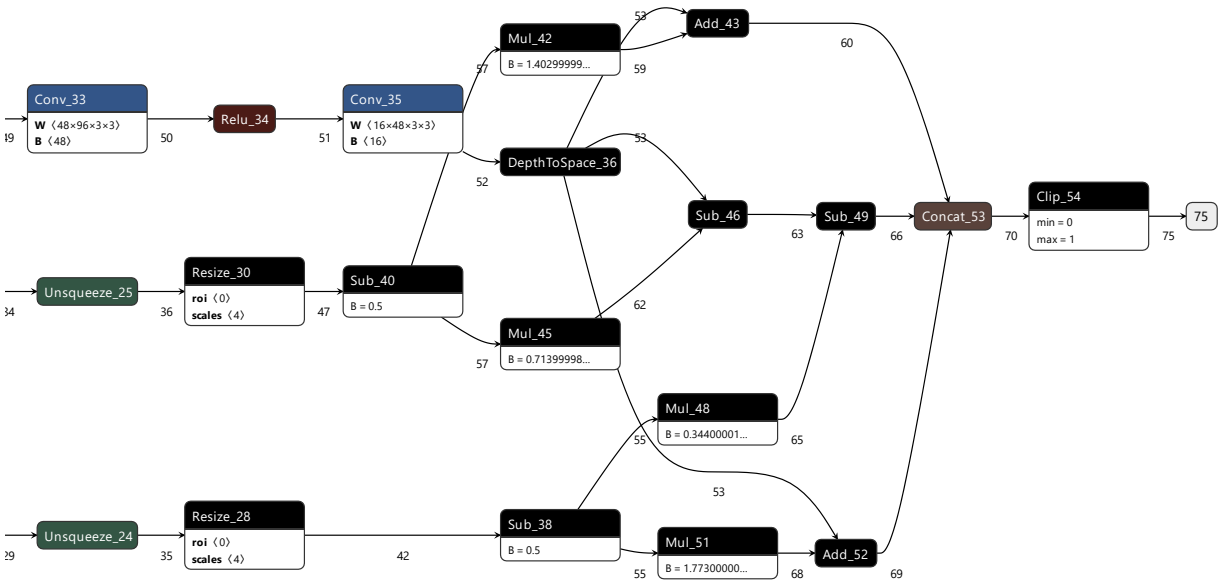
We visualize the model structure of ESPCN_RGB and ESPCN_YCbCr, using netron[11]. Netron can analyze ONNX model and visualize the structure of the model. First, the model structure of ESPCN_RGB is shown in Fig.1. There are only convolutions, ReLUs, clip function, and DepthToSpace (i.e. PixelShuffle). Next, the model structure of ESPCN_YCbCr is shown in Fig.2. We dividedly show the structure of ESPCN_YCbCr that is more complex than ESPCN_RGB since it has a function to convert color-space in itself. According to Fig. 2(a), ESPCN_YCbCr divides RGB signals to each color channel to calculate YCbCr channels. In addition, according to Fig.2(b), ESPCN_YCbCr concatenates

super-resolved luminance channel and algorithmically upsampled chroma channels. This conversion has not much computational complexity compared to three convolutions, but the calculation calls CUDA kernels multiple times, resulting in a bottleneck. In fact, the convolutions in ESPCN_YCbCr are not very computationally demanding, and the overhead of calling CUDA kernels multiple times by converting color-spaces cannot be ignored. To prove that the conversion is a bottleneck, we show the result of the analysis using Nsight Systems[12], that is a performance analysis tool to visualize CPU/GPU interworking. The result for ESPCN_RGB is shown in Fig. 3, and the result for ESPCN_YCbCr is shown in Fig. 4. It is confirmed that ESPCN_YCbCr calls CUDA kernel more than ESPCN_RGB and runtime per image is longer. In addition, we show Fig.5 to prove color-space conversion from YCbCr to RGB needs 2.5ms. For readability, we divided Fig 5. into four figures and show as Fig. 6. According to Fig. 6, color-space conversion is not light computation.

Fig.1 The structure of ESPCN_RGB



(a) first half



(b) second half

Fig.2 The structure of ESPCN_YCbCr

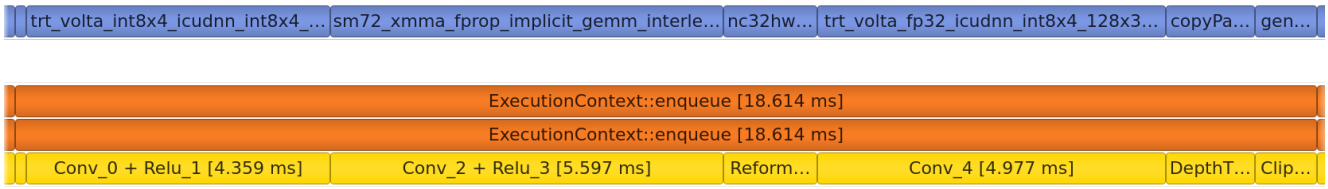


Fig.3 The result of the performance analysis of ESPCN_RGB

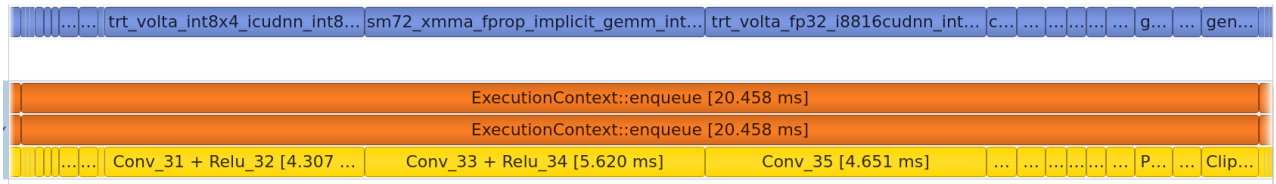


Fig.4 The result of the performance analysis of ESPCN_YCbCr

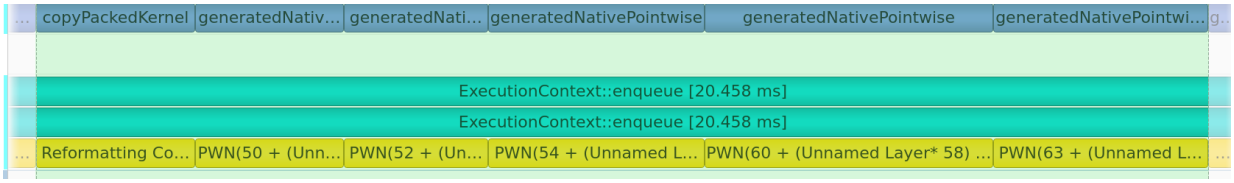
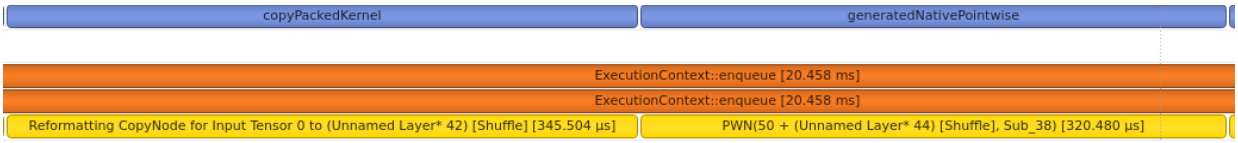
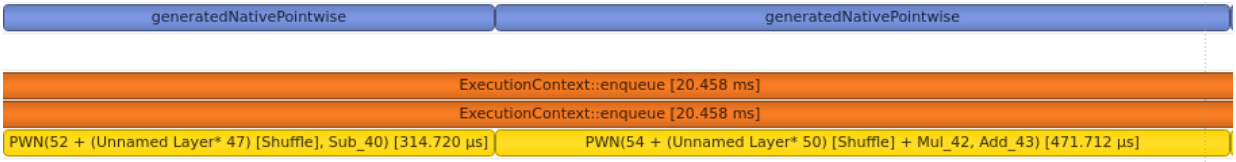


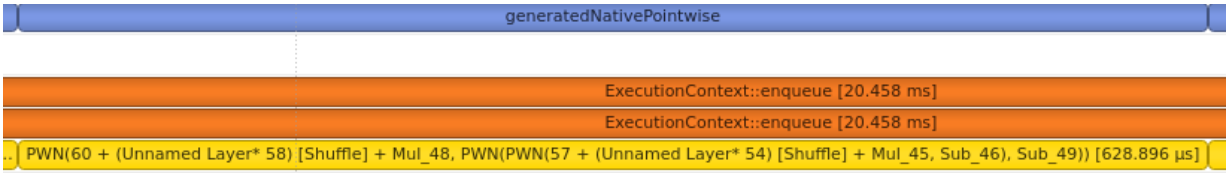
Fig.5 The runtime analysis of the conversion color-space from YCbCr to RGB



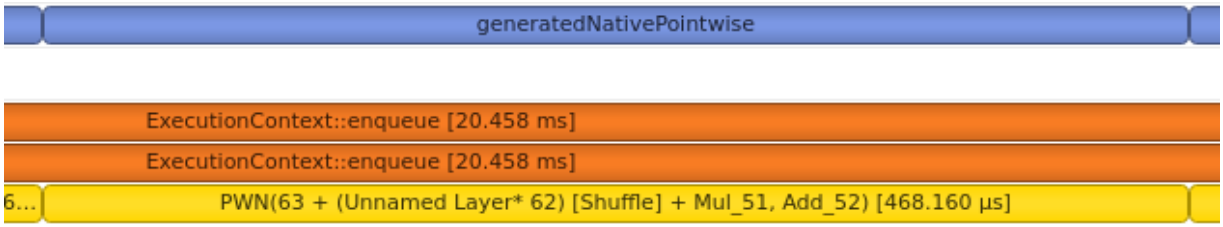
(a) first one



(b) second one



(c) third one



(d) fourth one

Fig.6 The detailed analysis of the runtime of the functions for converting color-space from YCbCr to RGB

Although ESPCN_YCbCr has little more computational cost, the PSNR is much better than ESPCN_RGB. Particularly, ESPCN_YCbCr has good performance when it is quantized. As shown in Table. 1, without calibration, quantized

ESPCN_RGB lost PSNR significantly (2.1415 [dB]) from pretrained. However, ESPCN_YCbCr lost PSNR less (0.7744 [dB]) from pretrained. From the viewpoint of the speed, we found that inference using TensorRT and INT8 precision is more than four times faster than inference using PyTorch's FP32 precision. This reason can be attributed to the Tensor core in Xavier NX, which is capable of performing 4x4x4 matrix products at high speed, and it is thought that TensorRT converts convolutional operations into matrix products and uses the acceleration provided by the Tensor core.

IV. Conclusions

In this paper, TensorRT was used to infer for the super-resolution model. Although conversion of the FP32 model to INT8 precision generated a decrease in PSNR, the calibration function was able to suppress the decrease of the accuracy. This is because TensorRT measures the distribution of the input tensor values when converting from floating-point to fixed-point and calculates how to scale the tensor to fit into the INT8 range and sets an appropriate dynamic range. We also found that converting the model to INT8 was more than four times faster than PyTorch's (FP32) inference. We can conclude that the quantization provided by TensorRT is useful in situations that require fast inferences.

In conclusion, we successfully implemented super-resolution running on mobile devices by 50 fps. As future work, we try to compare the automatic calibration with manual optimization of fixed point arithmetic, and incorporate other super-resolution models under tradeoff between runtime and super-resolution accuracy.

Acknowledgment

This work was supported in part by NICT, Grant Number 03801, Japan.

References

[1] Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang, “Image Super-Resolution Using Deep Convolutional Networks”, TPAMI 2015, May 2015.

[2] Wenzhe Shi, Jose Caballero, Ferenc Huszar , Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, Zehan Wang: “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network”, IEEE CVPR 2016, June 2016.

[3] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, Yun Fu, “Image Super-Resolution Using Very Deep Residual Channel Attention Networks”, ECCV 2018, September 2018.

[4] Computer Vision Laboratory, ETH Zurich: Accessed on Nov. 30, 2021. [Online] Available: https://ai-benchmark.com/workshops/mai/2021/

[5] Computer Vision Laboratory, ETH Zurich: Accessed on Nov. 30, 2021. [Online] Available: https://data.vision.ee.ethz.ch/cvl/ntire21/.

[6] NVIDIA: “NVIDIA TensorRT | NVIDIA Developer”, https://developer.nvidia.com/tensorrt.

[7] “NVIDIA-AI-IOT/torch2trt: An easy to use PyTorch to TensorRT converter”, Accessed on: Jan. 10, 2022. [Online]. Available: https://github.com/NVIDIA-AI-IOT/torch2trt.

[8] NVIDIA: “Jetson Xavier NX for embedding/edge systems | NVIDIA”, https://www.nvidia.com/ja-jp/autonomous-machines/embedded-systems/jetson-xavier-nx/.

[9] Xiph.org: “Xiph.org : Derf's Test Media Collection”, https://media.xiph.org/video/derf/.

[10] Agustsson, Eirikur and Timofte, Radu: “NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study”, CVPR workshops, July 2017.

[11] Lutz Roeder: Accessed on: Nov. 28, 2021. [Online]. Available: https://netron.app/.

[12] NVIDIA: “NVIDIA Nsight Systems | NVIDIA Developer”, https://developer.nvidia.com/nsight-systems.