The UNet and wUUNet models optimization for real-time computations on mobile phones

Bochkov Vladimir Sergeevich

Postgraduate

Kataeval Liliya Yurievna

Doctor of Physico-mathematical Sciences, Full Professor

Nizhny Novgorod State Technical University

Abstract. In the paper we provide an analysis of translation neural networks trained via Pytorch and Keras frameworks into optimum for mobile phones calculating structure using TFLite and appropriate techics of theoretical optimization of calculation

Keywords: GPU, quantization, floating point number, PyTorch, Keras, TFLite, dilated convolutions, double-reduced convolutional layers

Special attention is currently being paid to the problem of preventing natural disasters. Forest fires are the most common catastrophe causing serious damage to the Russian economy. It should be noted that the most often guilty in the occurrence of these phenomena is a person and careless handling of fire. Also, one cannot ignore the fact that our country has vast forests, and there is not enough professional personnel to monitor them. Therefore, the scientific community is faced with the task of developing automated means of protecting and counteracting forest fires. One of the promising areas of computational cybernetics is the development of new methods of detection and analysis behind flames. In case of forest fires, the supply of water to the flame without taking into account its structure is ineffective, therefore it is necessary to develop an algorithm for extracting the zones of fire vulnerability on video.

As part of a study in the field of choosing the best mobile software and hardware system for fire safety tasks [1], a modern android smartphone was chosen in view of its optimal photo capabilities for color rendering of the flame and the presence of a mobile video processor.

In order to qualitatively improve the results of flame detection, approaches to flame segmentation were applied using modern convolutional neural networks of the UNet class [2], shown in fig. 1, and its modification was developed for the problem of segmentation of objects of the same kind on the basis of wUUNet [3], which is two UNet blocks (binary and refining multiclass) closely interconnected by throughput connectors, shown in fig. 2. Objects of the same kind within the framework of our task are the contours of the flame, and the sign that distinguishes the types of flame from each other is the color (Yellow, orange and red). This division was made in order to follow the rule of extinguishing a fire in the lower zone of the flame with a temperature of 600K [4], which corresponds to the red color in the video.

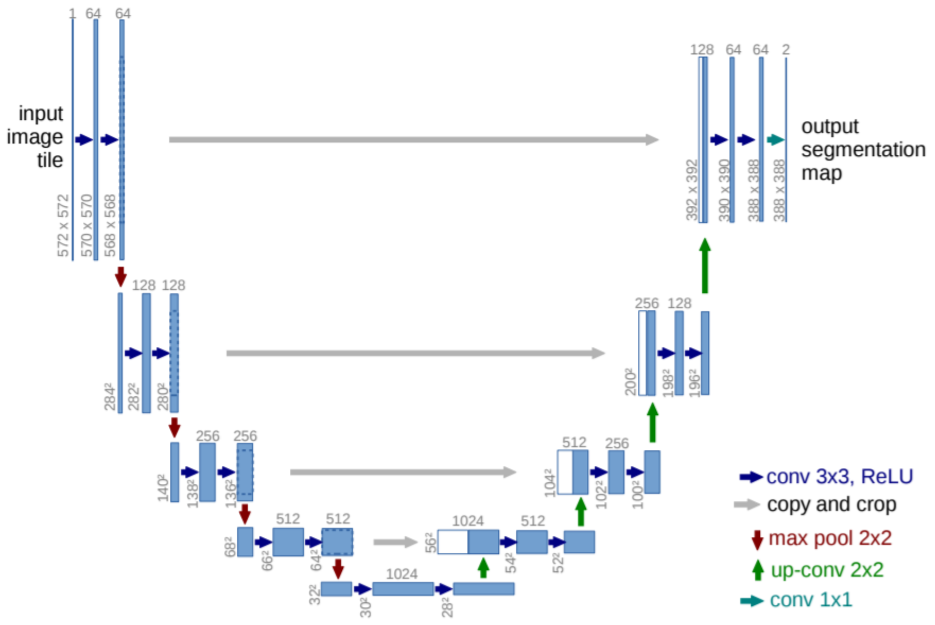


Fig. 1. – UNet neural network model architecture

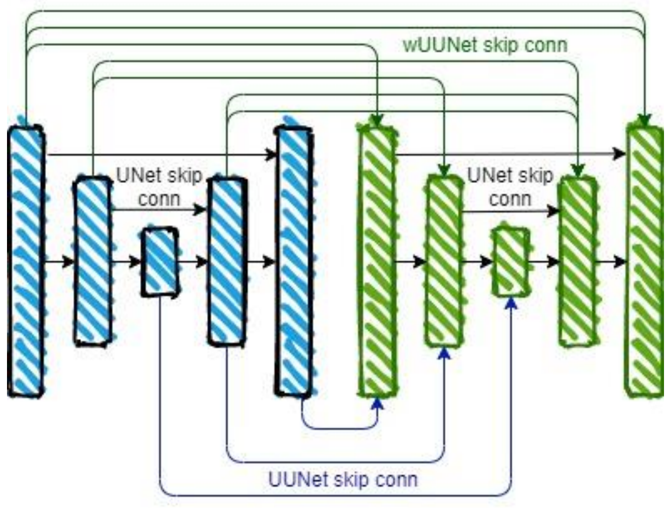


Fig 2. – Architecture of the wUUNet neural network model

Since mobile GPU computing uses the arithmetic of 16-bit floating point numbers [5], the quantization process, the rules of which are shown in fig. 3.the original float32 model needs to be configured for float16 calculations, which the TFLite library allows to do

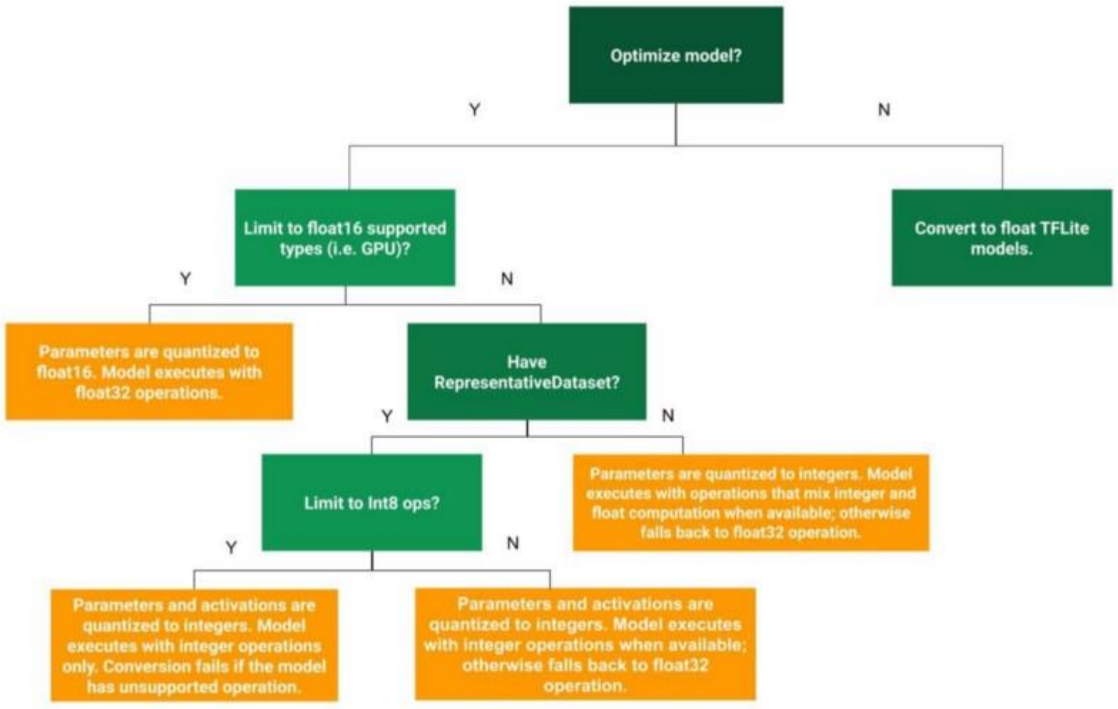

Optimize model?
Y
N
Limit to float16 supported types (i.e. GPU)?
Convert to float TFLite models.
Y
N
Parameters are quantized to float16. Model executes with float32 operations.
Have RepresentativeDataset?
Y
N
Limit to Int8 ops?
Parameters are quantized to integers. Model executes with operations that mix integer and float computation when available; otherwise falls back to float32 operation.
Y
N
Parameters and activations are quantized to integers. Model executes with integer operations only. Conversion fails if the model has unsupported operation.
Parameters and activations are quantized to integers. Model executes with integer operations when available; otherwise falls back to float32 operation.

Fig. 3 – Weights and Calculation Quantization Rules in TFLite Library

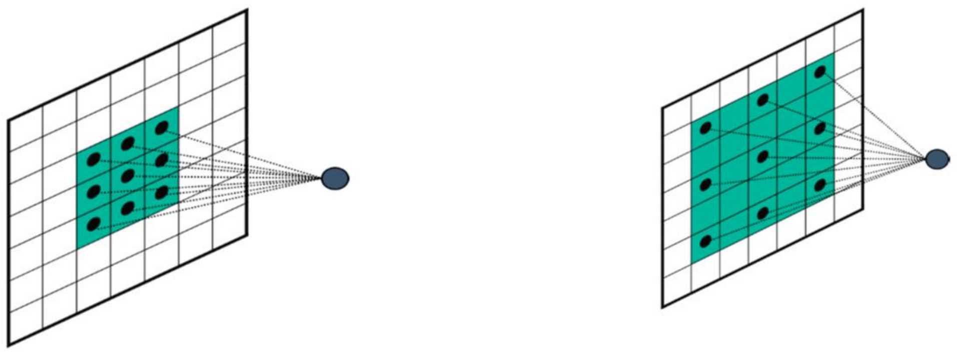
The first comparison involves UNet models developed on the PyTorch and Keras frameworks, the computation time of which is shown in tab. 1:

Model	Average time $\bar{t}$ ms, (FPS)	σ (std)	*min(t)*, ms. (FPS)	*max(t)*, ms. (FPS)
UNet Pytorch	180.58 (5.5 FPS)	**6.59**	147 (6.8 FPS)	217 (4.60)
UNet Keras	**158.49 (6.31 FPS)**	6.96	**128 (7.81 FPS)**	**181 (5.52 FPS)**

Tab. 1 – comparison of UNet models trained in Pytorch and Keras

From this table it can be seen that the model trained in Keras is faster on average than Pytorch. This is due to the fact that Keras uses the Tensorflow background related to the mobile computing library and the translation chain consists of the usual translation of the Keras model -> TF Lite, while in order to translate the Pytorch model to TF Lite, you need to go through several stages: Pytorch - > ONNX -> TensorFlow -> TF Lite, which contributes to sub-optimal model translation. Thus, it is optimal for mobile computing to use models trained on the Keras framework.

The following comparison is used between UNet models using sparse convolutions [6] and not. The thinning convolution technique is shown in fig. 4



(**a**) General convolution (kernel 3 × 3/rate = 1 (**b**) Dilated convolution (kernel 3 × 3/rate = 2)

Fig. 4 – Standard (a) and sparse convolutions (b)

From the theoretical point of view, the use of sparse convolutions can multiply the computational complexity, but in practice this is not the case, since GPU memory, like massively parallel computations, work efficiently only in the case of using continuous ranked memory access, and in the case of sparse convolutions, it is taken every second element horizontally and vertically. Comparative indicators are presented in tab. 2.

Model	Average time $\bar{t}$ ms, (FPS)	σ (std)	*min(t),* ms. (FPS)	*max(t),* ms. (FPS)
UNet Dilated	240.53 (4.16 FPS)	7.15	206 (4.85 FPS)	264 (3.78)
UNet	**158.49 (6.31 FPS)**	**6.96**	**128 (7.81 FPS)**	**181 (5.52 FPS)**

Tab. 2 – comparison of UNet models with and without sparse convolutions

The second way to optimize the model is to halve the convolutional layers: instead of 64,128,256,512, use 32,64,128,256 layer convolutions, respectively. Based on the tab. 3 this method effectively improves the execution time of the neural network:

Model	Average time $\bar{t}$ ms, (FPS)	σ (std)	*min(t),* ms. (FPS)	*max(t),* ms. (FPS)
wUUNet	240.53 (4.16 FPS)	7.15	206 (4.85 FPS)	264 (3.78)
UNet	158.49 (6.31 FPS)	6.96	128 (7.81 FPS)	181 (5.52 FPS)
wUUNet light	**147.96 (6,76 FPS)**	**6.88**	**123 (8.13 FPS)**	**168 (5.95 FPS)**
UNet light	**75.01 (13,33 FPS)**	**10.78**	**51 (19.60 FPS)**	**116 (8.62 FPS)**

Tab. 3 – comparison of UNet and wUUNet models in standard and light versions

Thus, it was found that it is effective to use a two-fold reduction in the layers of the neural network. This option allows even a dual wUUNet model to run faster on average than a standard UNet. The strategy of using sparse convolutions is not efficient in the context of computing on mobile processors.

References:

1. Bochkov V.S., Kataeva L.Yu. The expediency analysis of using android smartphone as an main module of water cannon control. Processing Management and Scientific Developments international conference. 2021, P. 147-152
2. Ronneberger O., Fisher P., Brox T. U-Net: Convolutional networks for biomedical image segmentation. arXiv:1505.04597 [cs.CV] 2015.
3. Bochkov V.S., Kataeva L.Y. WUUNet: Advanced fully convolutional neural network for multiclass fire segmentation. Symmetry, 2021, 13(1), P. 1–18, 98
4. Bochkov V.S., Belyaev I.V., Maslennikov D.A., Kataeva L.Yu., Iliicheva M.N. Analytical solution of the problem of combustion wave propagation in a homogeneous porous layer of organic combustible materials. ARPN Journal of Engineering and Applied Sciences, 2016, 11 (19), P. 11356-11362
5. Optimization Techniques – TFLite!! https://www.techwasti.com/optimization-techniques-tflite/
6. Fisher Yu, Koltun V. Multi-scale context aggregation by dilated convolutions. arXiv:1511.07122 [cs.CV] 2015.