MULTI-CLASS WEATHER CLASSIFICATION FROM SINGLE IMAGES WITH CONVOLUTIONAL NEURAL NETWORKS ON EMBEDDED HARDWARE

Tomáš Bravenec
Doctoral Degree Programme (2), FEEC BUT
E-mail: xbrave01@stud.feec.vutbr.cz

Supervised by: Tomáš Frýza
E-mail: fryza@vut.cz

Abstract: The paper is focused on creating a lightweight machine learning solution for classification of weather conditions from input images, that can process the input data in real time on embedded devices. The approach to the classification uses deep convolutional neural networks architecture with focus on lightweight design and fast inference, while providing high accuracy results. The focus on creating lightweight convolutional neural network architecture capable of classification of weather conditions also enables usage of the network in real time applications at the edge.

Keywords: deep learning, neural networks, computer vision, weather classification, machine learning, parallel computing, inference on edge, reduced precision computing

1 INTRODUCTION

The correct weather analysis is crucial for autonomous machines, as various weather conditions can disrupt the proper function of such machines. Current systems use mostly a sensor arrays or cameras to predict the weather conditions the machine is located in. This issue does not affect only autonomous machines but transportation infrastructure too. Also various traffic accidents could be prevented by better adjustment of driving to the current weather conditions with either not allowing to start autonomous driving system in cars, auto pilot in airplanes etc, requiring full attention from the driver or pilot while the autonomous systems are active or using different constraints for autonomous movement. As the weather can be quite diverse, the approach using classic computer vision techniques is not the best approach here, and the usage of deep neural networks is more suitable for learning the weather patterns directly out of training data.

As weather is not generally changing too quickly and latency is not an issue, it is possible to offload the computation from the device to servers in the cloud. There are situations the use of cloud computing is not an option, either due to lack of connectivity or privacy and security reasons. In situations like these, the inference has to be done on the same or closely connected device that captured the data. This process of doing the computation, usually on embedded devices close to the sensors acquiring the data is called a computing at the edge. For the usage of neural networks on edge devices, the neural networks are trained on a wide training set of data representing all the required situations either using a powerful server or desktop computer, due to the compute limitations of the embedded devices, this approach saves a lot of time as training neural networks on embedded hardware would be extremely time consuming and ineffective.

A lot of the related works use a binary classification, which means those solutions are capable of classifying only two weather conditions [1, 2], being either sunny or cloudy, sunny or raining etc. There are also works doing multi-class weather classification [3, 4] which mostly work with 3 classification labels, adding either snowing or foggy conditions into the classification.

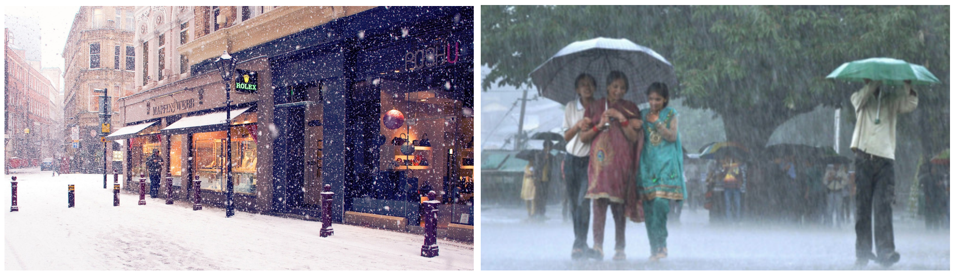

Figure 1: Example of images used for extension of the dataset containing rainy and snowy conditions

2 DATASET

The dataset used for training and testing the designed neural network architecture was created by modifying existing multi-class weather dataset [5], which contains 1530 images, with another 1189 images. The original dataset contained 5 weather conditions, each represented by 200 to 300 images. The weather conditions represented in the original dataset are cloudy, rainy, foggy, clear weather, sunrise or sunset.

The author's modification of the dataset was necessary as the original dataset did not include any images including snowy conditions, so the dataset was extended with a new class with 619 training images containing snowy weather. Example of snowy and rainy condition images used for the dataset expansion are shown in Figure 1. To make the dataset more robust, the original classes were also extended with about 100 to 200 additional images under Creative Commons licence scraped from the internet.

3 IMPLEMENTATION

Due to the requirement of real time processing on embedded devices, the neural network was designed and trained from scratch with embedded devices in mind. This means the deep neural network uses relatively low amount of layers with small amount of trainable parameters. Bigger networks with more parameters and layers were also tested but the result improvement at at most 1 % was not worth the increase in size and complexity of the network. The architecture of the designed network is in Table 1.

The training of the network was done using Quantization Aware Training (QAT) method, which adds another constraint to the weights of the network, to allow them to be converted from the standard 32-bit floating point format into 8-bit fixed point precision format without significant loss in accuracy of the network.

For the training itself the framework for deep learning applications TensorFlow 2 [6] was used, as it allows conversion of the trained model into TensorFlow Lite model. It is a highly optimized version of TensorFlow specifically designed to be ran in mobile applications and on embedded devices with low compute performance.

4 EVALUATION AND TESTING

The testing of the designed convolutional neural architecture was done on a single board computer NVIDIA Jetson Nano [7]. The 4-core CPU of NVIDIA Jetson Nano are built on an Arm A57 architecture and they are clocked at frequency of 1.5 GHz. The NVIDIA Jetson series of single board computers also contain desktop grade GPU cores even though in smaller numbers than their desktop counterparts. The GPU is based on NVIDIA Maxwell architecture, and contains 128 CUDA cores running at frequency of 0.9 GHz. The NVIDIA Jetson Nano contains 4 GB of DDR4 memory, which

Table 1: Designed convolutional neural network architecture

Layer	Input Shape	Output shape	Activation	Parameters
Convolutional 2D	224x224x3	222x222x16	ReLU	448
Max pooling	222x222x16	111x111x16	—	0
Convolutional 2D	111x111x16	109x109x8	ReLU	1160
Max pooling	109x109x8	54x54x8	—	0
Convolutional 2D	54x54x8	52x52x8	ReLU	584
Max pooling	52x52x8	26x26x8	—	0
Convolutional 2D	26x26x8	24x24x8	ReLU	584
Convolutional 2D	24x24x8	22x22x4	ReLU	292
Flatten	22x22x4	484x1	—	0
Fully connected	484x1	40x1	ReLU	19400
Fully connected	40x1	40x1	ReLU	1640
Fully connected	40x1	40x1	ReLU	1640
Fully connected	40x1	6x1	SoftMax	246

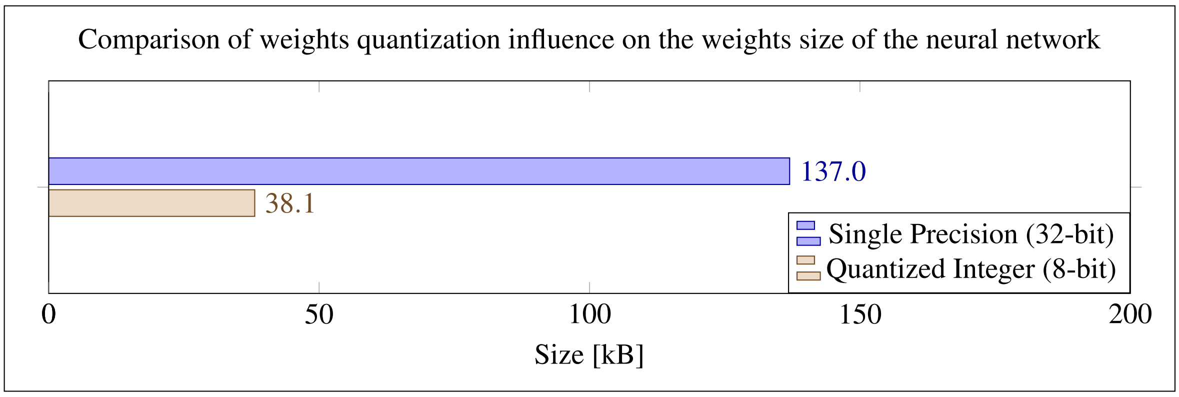

Figure 2: Comparison of data type influence on the size of weights

is dynamically assigned to either the CPU or the GPU. The whole system is relatively low power and consumes at most 10 W which makes it perfect compute system for applied computer vision projects.

Since the NVIDIA Jetson Nano has desktop class NVIDIA CUDA cores usable for general purpose computing and parallel data processing acceleration, but TensorFlow lite does not support it, the quantized model could not be tested on the Jetson Nano's GPU. The model using standard 32-bit floating point arithmetic was tested both using the CPU and the GPU.

When it comes to the size of the weights of the designed neural network, the usage of 8-bit quantized integer weights made the model a lot smaller than while using the standard 32-bit floating point data type, to be exact 3.6 × smaller, which is expected as almost every weight is quarter of the original size. The comparison of the weights size using reduced precision data type is in Figure 2 while the comparison of accuracy as a whole and inference performance using standard 32-bit floating point and quantized integer weights is in Figure 3.

4.1 Standard 32-bit arithmetic

From the accuracy point of view at the designed network, the results are quite reasonable for the small size and efficiency of the network. The correct weather conditions with the network using 32-bit

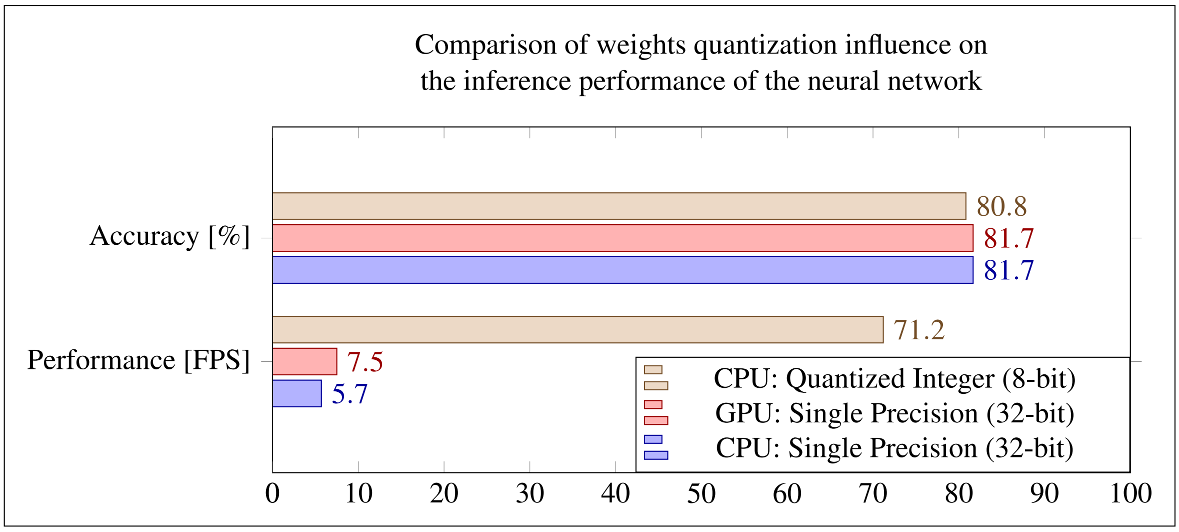

Figure 3: Comparison of data type influence on the inference performance

floating point arithmetic were predicted correctly in 81.67 % of all cases. The reason for almost 20 % incorrect classifications is the variability of weather. There can be clouds in the sky even on a sunny day, or snowflakes can be mistaken for rain drops and vice versa. The confusion matrix showing the ground truth and predictions of the designed network are in Figure 4 a).

The inference speed of the standard model was not great at only 5.7 frames per second using CPU, and 7.2 frames per second using GPU. The relatively small increase in performance using the GPU acceleration is due to the small size of the designed neural network. In this network, memory transfers (copy of the input data in and out of the memory space assigned to the GPU) take majority of the time required for the processing which in the end does not give that big performance improvement.

4.2 Quantized 8-bit arithmetic

The network using quantized 8-bit arithmetic, provides very similar results in terms of accuracy to the non quantized network at 80.83 % of all predictions being correct, which is just 0.84 % worse than using the non-quantized model. The confusion matrix of the quantized network is in Figure 4 b).

The performance gained by converting the model using TensorFlow Lite to use quantized integer arithmetic provided massive bump in inference performance, by managing to get frame rate of 71.2 frames per second which is a lot higher than required for real-time processing.

5 CONCLUSION

The main focus of this paper was to design a system for weather classification, that would be simple and lightweight enough to run on an embedded hardware. For this the training dataset was expanded to contain almost double the amount of images. The designed neural network created using TensorFlow is small and provides accurate results. While using traditional 32-bit floating point arithmetic, the performance of the designed neural network was not great, even while using the integrated GPU of the NVIDIA Jetson Nano, managing at most 7.5 frames per second. The usage of weights quantization and fixed point arithmetic with TensorFlow Lite provided a lot better results as the neural network managed to achieve better than real-time inference performance at 71.2 frames per second. The designed neural network could possibly be expanded in the future to be capable of classification of more weather conditions, like thunderstorm, sandstorm, hurricanes and more.

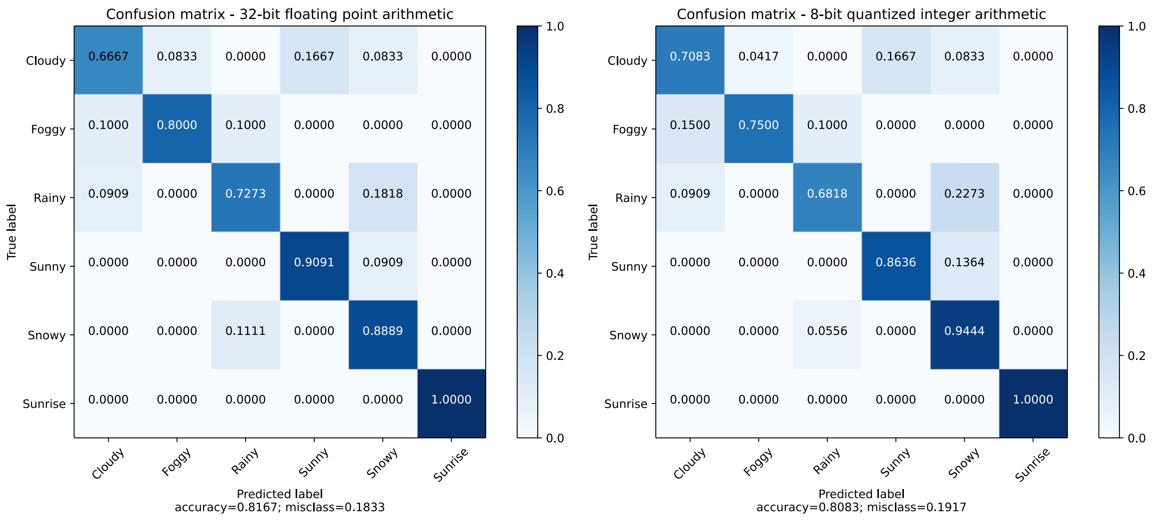

Figure 4: Confusion matrix of weather classification CNN: a) with 32-bit floating point arithmetic, b) with 8-bit quantized integer arithmetic

ACKNOWLEDGEMENT

The research published in this submission was financially supported by the Brno University of Technology Internal Grant Agency under project no. FEKT-S-20-6325.

REFERENCES

[1] M. Elhoseiny, S. Huang, and A. Elgammal, “Weather classification with deep convolutional neural networks,” in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 3349–3353.

[2] C. Lu, D. Lin, J. Jia, and C. Tang, “Two-class weather classification,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2510–2524, 2017.

[3] Z. Zhang and H. Ma, “Multi-class weather classification on single images,” in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 4396–4400.

[4] S. Jabeen, A. Malkana, A. Farooq, and U. Ghani Khan, “Weather classification on roads for drivers assistance using deep transferred features,” in *2019 International Conference on Frontiers of Information Technology (FIT)*, 2019, pp. 221–2215.

[5] V. Gupta, “Weather dataset,” https://www.kaggle.com/vijaygiitk/multiclass-weather-dataset, 2020, [Online; accessed 15-February-2021].

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[7] Embedded Linux Wiki , “Jetson,” https://elinux.org/index.php?title=Jetson&oldid=543051, 2021, [Online; accessed 29-January-2021].