Deep learning inference in GNU Radio with ONNX

Oscar Rodriguez OSCAR.RODRIGUEZZALONA@HEIG-VD.CH
Alberto Dassatti ALBERTO.DASSATTI@HEIG-VD.CH
School of Management and Engineering Vaud (HEIG-VD), University of Applied Science Western Switzerland (HES-SO), Yverdon-les-Bains, Switzerland

Abstract

This paper introduces gr-dnn [1], an open source GNU Radio Out Of Tree (OOT) block capable of running deep learning inference inside GNU Radio flow graphs. This module integrates a deep learning inference engine from the Open Neural Network Exchange (ONNX) project. Thanks to the interoperability with most of the major deep learning frameworks, it does not impose any restriction on the tool used by the model designer. As an example, we demonstrate here its functionalities running a simple deep learning inference model on raw radio samples acquired with a PlutoSDR.

1. Introduction

Deep Learning (DL) is a branch of Machine Learning (ML) that has been used successfully to solve complex problems in different domains like image processing (Krizhevsky et al., 2012), natural language processing (Collobert & Weston, 2008), and speech recognition (rahman Mohamed et al., 2012). In recent years, deep learning techniques have been also applied to wireless communication and spectrum sensing (Clancy et al., 2007; Hossain et al., 2014; Arjoune & Kaabouch, 2019) showing that they can outperform classical approaches in tasks like automatic modulation classification (O'Shea et al., 2018). Integrating such technology in a Software Defined Radio (SDR) framework seems very promising and could lead to new ideas in the field, for instance in cognitive radio and signal classification.

We think that the best way to integrate DL and SDR is interfacing a ML framework inside GNU Radio. There are many DL frameworks available nowadays that allow creating, training, and running deep learning models. Design and train of DL models are normally executed in a specific development environment and we only focus here on running inference inside GNU Radio. Most ML frameworks and inference engines are often incompatible with each other and a model created and trained with one framework can not be easily manipulated in a different one. The same is the norm form inference engines: compatibility is a real big issue. A second relevant aspect is performance. Deep learning inference is a computationally heavy task and the amount of data acquired in an SDR context can be huge: It is important to have access to optimizations and modern hardware. Different deep learning frameworks provide different optimization techniques and support for different hardware acceleration platforms.

[1]https://gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn

In the next section 2, we discuss about the inference engine selected and its capabilities. In section 3, we describe how gr-dnn works and how to configure it. Then in section 4 we explain how to use our block in a real scenario using a convolutional neural network (CNN) trained for modulation classification (O'Shea & Hoydis, 2017). Finally, in section 5 we present the results obtained.

2. ONNX Runtime inference engine

ONNX Runtime (Microsoft, b) is an inference engine that supports models based on the ONNX format (Microsoft, a). ONNX is an open format built to represent machine learning models that focuses mainly on framework interoperability. It defines a common set of operators used to create ML and DL models and a file format that enables using these models in a different frameworks. In practice, this means that each developer can use their preferred ML framework without worrying about downstream future implications. It also supports hardware acceleration for several platforms including GPUs (and partially FPGAs [2]).

ONNX format is supported by multiple frameworks either natively by the framework or through third party converter tools (ONNX, 2020).

In summary, ONNX Runtime provides an inference engine

[2]https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service

with support for models trained with a variety of frameworks and at the same time it takes advantage of specific hardware accelerators when available. It seems a natural choice considering the constraints presented in section 1.

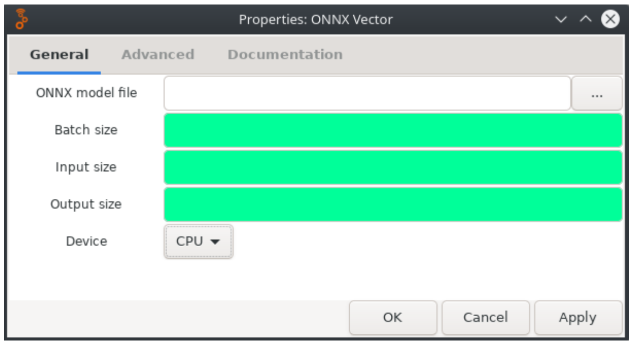
3. gr-dnn

In order to integrate a deep learning solution in an SDR context, we have developed a GNU Radio OOT module (Radio) called **gr-dnn** that allows us to use ONNX Runtime deep learning inference engine inside GNU Radio flow graphs.

Our block consists of a synchronous block [3] (it consumes and produces an equal number of items per port) developed in Python. It makes use of the Python bindings of ONNX Runtime to read the metadata of the loaded ONNX model and configure the ONNX Runtime backend with the correct parameters. Then, we adapt (reshape in row-major order (Wikipedia contributors, 2020b)) the input of the block to the expected format for the input of the model. Finally, we take the output of the model, adapt it (flatten, more details about this con be found in section 4) and write it to the output port of our block.

3.1. Block configuration

The configuration of a gr-dnn block consists of the following parameters:



ONNX model file	
Batch size	
Input size	
Output size	
Device	CPU

Figure 1. gr-dnn configuration

- **ONNX model file** is the ML trained model file in ONNX format.
- **Batch size** is the number of inputs that will be used for inference at once. This is useful to alleviate some of the overhead produced by moving data, for instance, from the capturing device to the device where the inference will be executed.
- **Input size** is used to define the size of the input port of the block (for consistency with other blocks) and it has to match with the size of a flatten input of the loaded model.
- **Output size** is used to define the size of the of output port of the block (for consistency with other blocks) and it has to match with the size of a flatten output of the loaded model.
- **Device** is the device that will be used for inference. If selected device is not available, the default device will be used (most of the cases that is CPU).

4. Experimental setup

We have prepared a flowgraph [4] that implements one possible use case for deep learning: signal classification. It processes raw I/Q samples acquired by a PlutoSDR (figure 2). In this scenario, we are going to use a CNN model for automatic modulation classification (O'Shea & Hoydis, 2017) (model layout and complexity can be found in the original article). The input of the model is a vector of 128 I/Q pairs values as single-precision float-values and the output is a vector with the probability of the samples being encoded in one of the ten different digital and analog single-carrier modulation schemes used during model training [5] (8PSK, AM-DSB, BPSK, CPFSK, GFSK, PAM4, QAM16, QAM64, QPSK and WBFM).

The input of the block has to be a flattened tensor. For example, if the model expects a 3D tensor as input, like an image, with dimensions $(64, 64, 3)$ the first two corresponds to the width and height of the image and the last dimension is the number of channels (e.g. RGB) then, the block would expect a vector with size $64 \times 64 \times 3$ representing the flatten tensor. In our case, the model expects a 2D tensor with dimension $(128, 2)$ where the first dimension is the number of samples and the second dimension represents I and Q values, then the input port of the block should have a size of 256 (128×2).

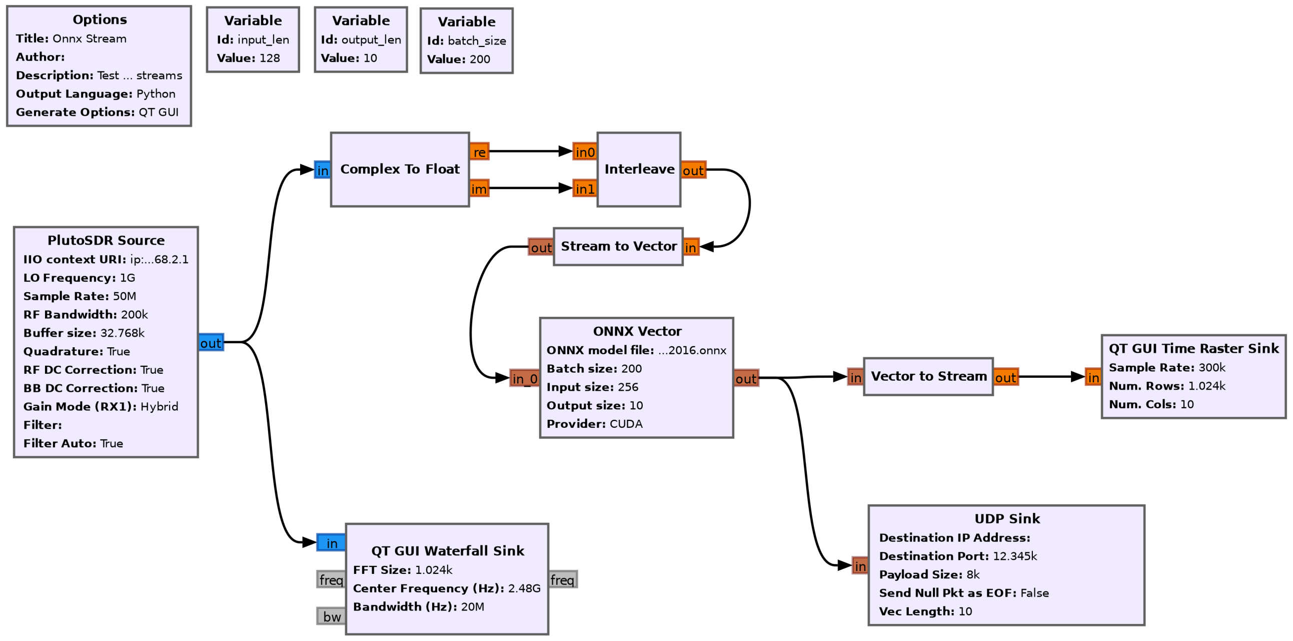
For simplicity, in this demo, we have not applied any pre-processing to the data. It is relevant to remember that any pre-processing method used during training has to be applied during inference as well. Notable examples are normalization or standardization (Wikipedia contributors, 2020a)

The output of the model is a flattened tensor as well. It will contain the probability of every of the possible modu-

[3]https://wiki.gnuradio.org/index.php/Types_of_Blocks

[4]https://gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn/-/blob/master/examples/pluto_onnx_vector.grc

[5]RNK2016.10b - https://www.deepsig.ai/datasets



Options
Title: Onnx Stream
Author:
Description: Test ... streams
Output Language: Python
Generate Options: QT GUI

Variable
Id: input_len
Value: 128

Variable
Id: output_len
Value: 10

Variable
Id: batch_size
Value: 200

PlutoSDR Source
IIO context URI: ip:...68.2.1
LO Frequency: 1G
Sample Rate: 50M
RF Bandwidth: 200k
Buffer size: 32.768k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Hybrid
Filter:
Filter Auto: True

ONNX Vector
ONNX model file: ...2016.onnx
Batch size: 200
Input size: 256
Output size: 10
Provider: CUDA

QT GUI Time Raster Sink
Sample Rate: 300k
Num. Rows: 1.024k
Num. Cols: 10

QT GUI Waterfall Sink
FFT Size: 1.024k
Center Frequency (Hz): 2.48G
Bandwidth (Hz): 20M

UDP Sink
Destination IP Address:
Destination Port: 12.345k
Payload Size: 8k
Send Null Pkt as EOF: False
Vec Length: 10

Figure 2. Flowgraph for deep learning inference with PlutoSDR and data preparation.

lations for the input sequence. Then, we repurpose a plotting tool from GNU Radio (time raster sink) to show the output probabilities of the classification as an easy way to visualize the classification results.

If we need a better representation of the results of the classification we have to use externals tools. In this case, for a real time visualization of the classification, we propose to send the output of our block (flattened tensor of the output of the model) to a UDP Sink.

Then, we can use a simple python script to read the information from the UDP socket and then plot the output of our block. Because we are sending the result of a batch we plot the average probability for each modulation in the batch in a bar plot (figure 3)

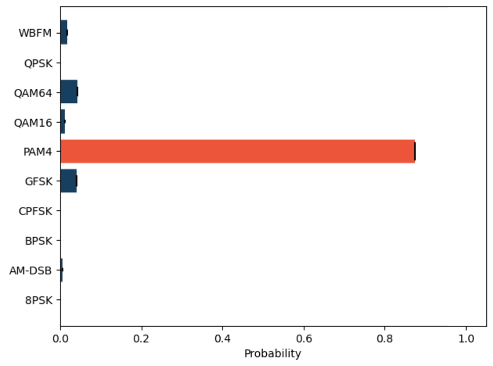


Figure 3. Representation of the output using an external tool

5. Results

In order to demonstrate the deep learning classification using **gr-dnn**, we prepared some data visualization in the flow graph (figure 4). We plot the raw data used as input in a waterfall sink (top) and the output of the classification in a time raster sink (bottom). The number of columns on the time raster sink corresponds to the size of the output of the model which, for classification, is the probability of belonging to each of the possible classes. Each row in the raster sink corresponds to one output of the model. This is just an example that helps us to visualize the input and output of the deep learning model in our block.

Throughput clearly depends on the model complexity: More complex models require higher computation. We have compared the performance running inference with different configurations (table 1): different sources, difference hardware (an Intel i7-7700K CPU and an NVIDIA RTX 2080 Ti GPU) and different batch sizes.

In tablet 1, we can observe the effect of the batch size on the performance. Using a batch size of 1 there is no difference with different hardware or source. If we increase the batch size, we start to see how the CPU limits the inference and perform worse than the GPU. For an even bigger batch size, we find a more significant performance gap between GPU and the CPU (more than x25 performance gain). If

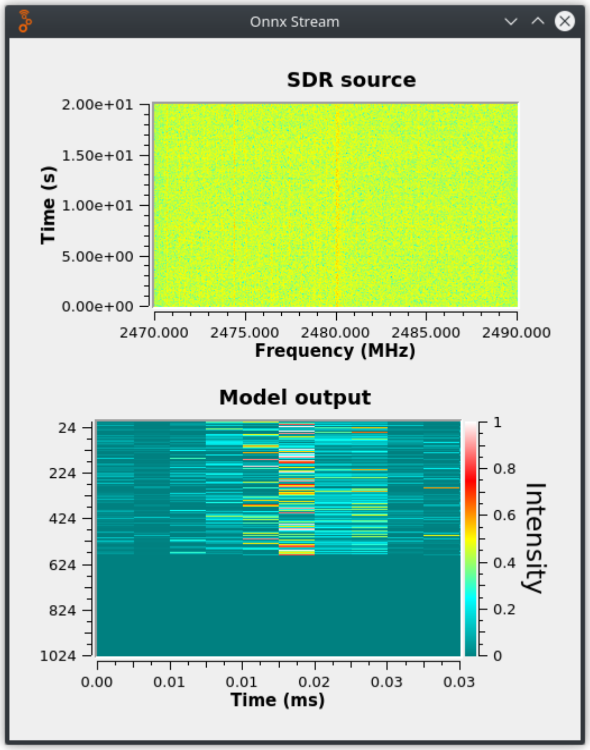


Figure 4. Output of figure 2 flow graph.

we compare the GPU results for different sources at batch size 100 we can observe a substantial difference that might be a bottleneck due to the USB 2.0 used between the SDR device and the computer. From these results, we can anticipate that we can run real-time inference if we have hardware acceleration but we will have a limited performance without it.

6. Conclusion

We have developed an GNU Radio OOT that integrates a deep learning inference engine: **gr-dnn**. It allows you to use deep learning solutions seamlessly inside GNU Radio. It supports multiple deep learning frameworks and hardware accelerations. To demonstrate the potential of this OOT module, we have shown how easy is to develop a fully functional example of deep learning inference for automatic modulation classification using raw I/Q values from an SDR device, and running it in real-time.

Similar solutions, like gr-wavelearner from Deepwave Digital [6], are less flexible because they require a specific hardware or they are tied to a specific platform. Both ONNX and ONNX runtime are abstraction layers, first for ML model declaration and second for ML inference.

Source	Hardware	Batch size	Bandwidth
File	CPU	1	3.5 MB/s
File	CPU	10	6.4 MB/s
File	CPU	100	7.1 MB/s
File	GPU	1	3.3 MB/s
File	GPU	10	30.1 MB/s
File	GPU	100	196.1 MB/s
File	GPU	1000	408.6 MB/s
SDR	CPU	1	3.3 MB/s
SDR	CPU	10	6.6 MB/s
SDR	CPU	100	7.3 MB/s
SDR	GPU	1	3.3 MB/s
SDR	GPU	10	30.2 MB/s
SDR	GPU	100	37.3 MB/s

Table 1. Block performance in different scenarios

All files (included examples and test models) are available in the repository of the project [7]. We also provide docker images, with all required dependencies, for both CPU and GPU inference [8] to facilitate testing. We hope to be able to add FPGA support in the next future. For further information, you can check the wiki of the project [9].

Acknowledgements

This work was partly funded by the European Space Agency (ESA) as part of the initiative SDR Makerspace. For more information you can visit the web page of the initiative [10].

References

Arjoune, Youness and Kaabouch, Naima. A Comprehensive Survey on Spectrum Sensing in Cognitive Radio Networks: Recent Advances, New Challenges, and Future Research Directions. *Sensors*, 19 (1):126, jan 2019. ISSN 1424-8220. doi: 10.3390/s19010126. URL https://www.mdpi.com/1424-8220/19/1/126.

Clancy, T., Hecker, Joe, Stuntebeck, Erich, and O'Shea, Tim. Applications of machine learning to cognitive radio

[6] https://github.com/deepwavedigital/gr-wavelearner

[7] https://gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn

[8] registry.gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn/gnuradio:[onnx$||onnx-cuda$]

[9] https://gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn/wikis

[10] https://sdrmaker.space

networks. *Wireless Communications, IEEE*, 14:47 – 52, 09 2007. doi: 10.1109/MWC.2007.4300983.

Collobert, Ronan and Weston, Jason. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

Hossain, Peter, Komisarczuk, Adaulfo, Pawetczak, Garin, Dijk, Sarah Van, and Axelsen, Isabella. Machine learning techniques in cognitive radio networks, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Microsoft. Onnx open neural network exchange. `https://onnx.ai/`, a.

Microsoft. Onnx runtime. `https://microsoft.github.io/onnxruntime/`, b.

ONNX. Onnx converting tools. `https://github.com/onnx/tutorials#converting-to-onnx-format`, 2020.

O'Shea, Timothy J. and Hoydis, Jakob. An Introduction to Deep Learning for the Physical Layer. *CoRR*, abs/1702.0, feb 2017. URL `http://arxiv.org/abs/1702.00832`.

O'Shea, Timothy James, Roy, Tamoghna, and Clancy, T. Charles. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, Feb 2018. ISSN 1941-0484. doi: 10.1109/jstsp.2018.2797022.

Radio, GNU. Gnu radio out-of-tree module. `https://wiki.gnuradio.org/index.php/OutOfTreeModules`.

rahman Mohamed, Abdel, Dahl, G., and Hinton, Geoffrey E. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20:14–22, 2012.

Wikipedia contributors. Normalization (statistics) — Wikipedia, the free encyclopedia, 2020a. URL `https://en.wikipedia.org/wiki/Normalization_(statistics)`.

Wikipedia contributors. Row- and column-major order — Wikipedia, the free encyclopedia, 2020b. URL `https://en.wikipedia.org/wiki/Row-_and_column-major_order`.