



CERN Summer Student Programme 2021

Tests and Benchmarks for TMVA/SOFIE

Final report

Federico Sossai

Department EP-SFT (Software Development for Experiments)

Period June 14th → September 10th

Supervisors Lorenzo Moneta, Sitong An



ABSTRACT

TMVA/SOFIE is a new experimental module part of the ROOT project that is intended to convert pre-trained deep neural networks into a C++ header file that carries minimal dependencies on external libraries.

This report is intended to summarize the work that I did as a participant to the CERN Online Summer Student Programme 2021, held remotely. The outcome of this work is code, as this is mainly a software project. I organized a CMake solution to allow quick and easy integration of GoogleTests and GoogleBenchmarks into the SOFIE project.

I also developed a class that allows the user to profile an inference task in SOFIE via code instrumentation. This is helpful in identifying the computational bottlenecks of the generated models.

INTRODUCTION

[ROOT](#) is an open source project begun in the nineties at CERN for analyzing and plotting big amounts of data. From a technical perspective it is divided into modules each one intended to provide a set of useful features, such as reading and writing big files, compressing, plotting or fitting functions just to outline a few. Among these we find [TMVA](#) (Toolkit for Multivariate Analysis), a module of ROOT that provides a rich set of machine learning tools like neural networks, boosted decision trees and other classifiers for both binary and multiclass problems.

Like many other fields, high energy physics has been deeply influenced by neural networks. In fact, there are experiments at LHC that use neural networks to quickly select whether an event is a proton-proton collision or more generally, if an event is potentially interesting and must be collected.

Machine learning at CERN cannot afford slow inference times as the accelerators produce up to 1 billion collisions per second, and if hardware and software systems are not prepared to work at a certain level of performance, precious events may get lost because of a too large latency.

My contributions revolve around the experimental TMVA/SOFIE, a new module of TMVA under active development.

WHAT IS SOFIE

A wide number of alternatives are available when the goal is training a neural model, but when it comes to deploying it, the available solutions are cumbersome or require too many dependencies to be carried into the production environment.

This is where SOFIE comes into play.

[SOFIE](#) (System for Optimized Fast Inference code Emit) is an experimental module of TMVA that aims at bridging the gap between a complete environment for training models and a fast environment for running them. SOFIE is designed to act like a compiler that *takes machine learning models as input and produces a C++11 header file that can be plugged in any project and used to carry out inference tasks.*

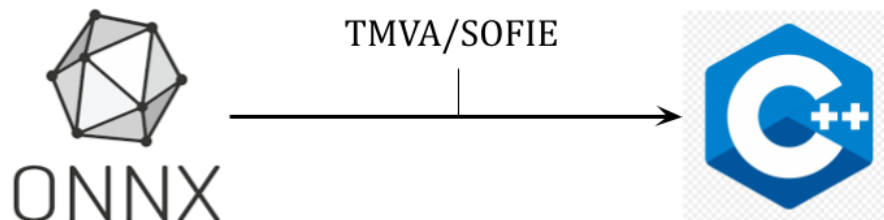


Fig.1: SOFIE can convert .onnx files to C++ headers.

It is recalled that SOFIE is still under development and not every ONNX operator is supported yet. At the moment of writing the supported operators are: Gemm (matrix multiplication), Conv (for 2D convolutions), ReLU,

In order to understand how SOFIE can process a machine learning model it is necessary to introduce how the latter is encoded.

THE ONNX FORMAT

Standardizing the representation of information is the ultimate goal for allowing a comfortable exchange of data between people working on the same concepts. The [ONNX](#) format does this: it was born to deliver to the machine learning community an open source common representation for neural networks, hence the acronym Open Neural Network Exchange.

An ONNX file is a representation of a computational graph where each node is an operator that applies some function to an input tensor producing another one.

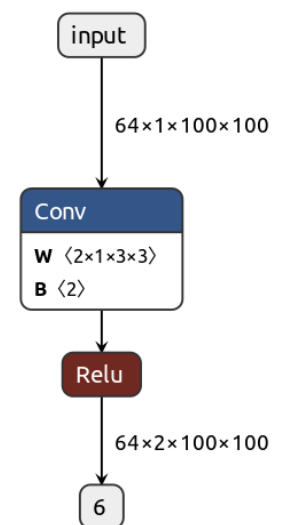


Fig.2: A shallow CNN stored in an ONNX file.

Figure 2 represents a convolutional neural network with a single convolutional operator and a ReLU function (image generated with [Netron](#)).

This format has been developed (and it is still subject to continuous improvement) by means of a Protobuf specification. [Protobuf](#) (short for Protocol Buffers) is a Google open source tool for serializing data to files. We can therefore say that ONNX is actually a Protobuf description of neural network models and this aspect is crucial when developing custom parsers for ONNX.

PARSING AN ONNX FILE

The ONNX format has been designed so that parsing a file in this format can be facilitated by ad-hoc APIs generated automatically by Protobuf targeting the programming language that the user wants to use.

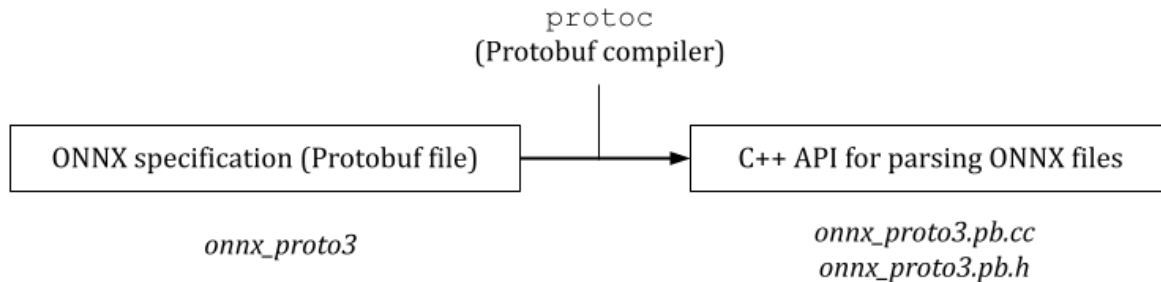


Fig.3: APIs for working with ONNX files are generated by the Protobuf compiler.

SOFIE makes extensive use of the generated API because it needs to inspect an input ONNX file identifying the computational nodes to be transformed into equivalent hardcoded C++ routines. Operators are created internally with a factory method. The following is an example of how a generated ReLU operator looks like:

```
for (int id = 0; id < 800; id++) {  
    tensor_22[id] = ((tensor_21[id] > 0) ? tensor_21[id] : 0);  
}
```

Fig.4: Example of a ReLU operator emitted by SOFIE.

MICROSOFT ONNX RUNTIME

Of course, there exist alternatives to running ONNX models. Microsoft developed a complete solution for running ONNX files, among other features. [ONNX Runtime](#) (ORT) is an

open source framework and represents the principal option for those who want to perform fast inference on CPUs or GPUs.

Our benchmark experiments will be compared against ORT since it represents the state-of-the-art solution for processing ONNX models.

TASK 1: ADDING GOOGLE TESTS

The process of translating models is very delicate because the main source code produces other source code that must perfectly reproduce every machine learning operator in every detail. Every, even tiny, implementation difference accumulates and propagates to the output accounting for a wrong inference result, invalidating correctness.

Given the importance of correctness, adding automated tests to SOFIE paves the way for a solid developing environment. For homogeneity with the rest of ROOT code, I developed these tests using [GoogleTest](#) and ad-hoc CMake scripts.

GTests for SOFIE are end-to-end: given a ONNX model composed of operators of which we want to assess the correctness of the implementation, the C++ header file is emitted, compiled, and run on a specified tensor input (generally an all-ones tensor). At this point the result is compared to a precomputed tensor, obtained from ORT. Any deviation beyond a certain threshold will cause the test to fail.

GTests are added and then generated according to the following workflow:

- A C++ file with an *expected output* of a given model must be created. The test will use it as a reference for assessing correctness.
- CMake is instructed to look for .onnx files in a predetermined directory.
- For every .onnx file, CMake instructs a Makefile (or any other build helper) to create the hardcoded header file by invoking a small SOFIE command line tool.
- Programmer is required to add a test case for each desider model.

This last step is not automatic, as there must be manual intervention to add the case in the canonical format (e.g. `TEST(ONNX, Linear16)`), but on the other hand, the programmer is not required to worry about the *compilation* of the new model. It will happen at make-time.

Note: It is important to specify in CMake that the target responsible for emitting the SOFIE C++ headers must be carried out **before** the beginning of GTest target as the latter directly depends on the emitted files.

TASK 2: PROFILING OPERATORS

Since SOFIE is designed for performance, among other things, I had the idea of developing a method for inspecting a hardcoded model beyond what can be achieved with a regular end-to-end benchmark. I developed a class called `RModelProfiler` that allows users to emit C++ headers that track every operators' time during any inference process.

This feature is implemented through *code instrumentation*, that is, by injecting dedicated code in between operators that store time points so that the time each operator took can be easily obtained through time point differences.

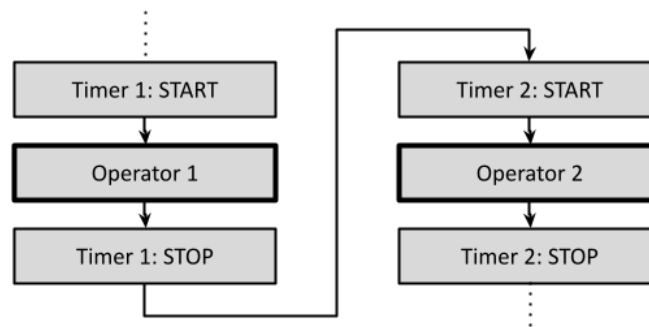


Fig.5: A header file emitted by SOFIE can be seen as a sequence of hardcoded operators each one of them having a clear beginning and end, this is why we can clock the computation with timers in between operators.

The following is a snippet taken from a header emitted with `RModelProfiler`.

```
tp_start = steady_clock::now();

for (int id = 0; id < 800; id++) {
    tensor_22[id] = ((tensor_21[id] > 0) ? tensor_21[id] : 0);
}

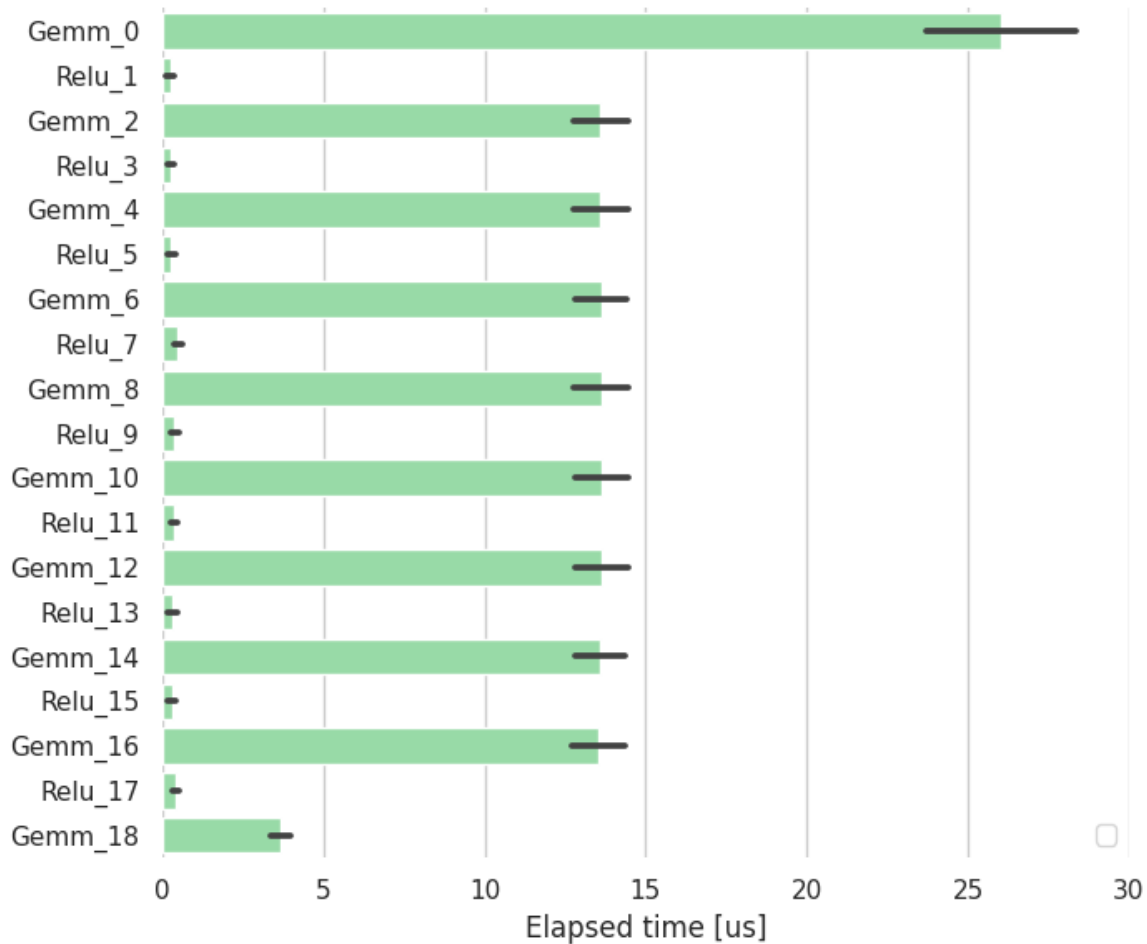
tp_stop = steady_clock::now();
current_execution["Relu_1"] =
    duration_cast<microseconds>(tp_stop - tp_start).count() / 1e0;
```

Fig.6: Example of how timers are implemented and how time is stored. ReLU operator.

Timings are stored in an unordered map using the op's name as a key so that the user can easily access the tracked data. As a helper feature, it is possible to get the *average execution time* and *variance* of every operator during multiple inference tasks.

Figure 7 represents the profiling of operators in a linear model composed of matrix

multiplications and ReLU activations. Inspecting a model to this level of detail is helpful in identifying performance bottlenecks or understanding where an optimization effort would not be beneficial.



*Fig.7: Plot of the profiling results obtained from a Multi-layer perceptron inference.
Standard deviation in black. 10'000 runs.*

We easily note three dominant behaviors in Gemms, in fact Gemm_0 is a [64,100]x[100,50] multiplication, Gemm_18 is a [64,50]x[50,10] multiplication and all the others are [64,50]x[50,50], hence the time differences.

TASK 3: BENCHMARKS - SOFIE VS ONNXRUNTIME

Among the [ROOT Project](#) we find [rootbench](#), the repository hosting benchmarks for all relevant functionalities of ROOT by means of [GoogleBenchmarks](#). End-to-end GoogleBenchmarks were still missing from rootbench, therefore I developed a CMake infrastructure for generating GoogleBenchmarks in a semi-automatic fashion.

It is in our interest to compare SOFIE emitted header performance against ORT library, because the comparison provides relevant feedback for determining whether SOFIE is competitive.

Files for GBench are located at [rootbench/root/tmva/sofie](#), where I designed two templated C++ file targeting ORT (`ONNXRuntimeInference_Template.cxx.in`) and SOFIE (`SOFIEInference_Template.cxx.in`). These two are not directly compilable since they are designed to be configured by CMake depending on which `.onnx` (for ORT) or `.hxx` (for SOFIE) files are to be benchmarked.

In order to add a new ORT GBench it is sufficient to place the `.onnx` file in [input_models](#): CMake will automatically discover new files and create the so-called *benchmark capture* for each one.

On the other hand, new SOFIE GBenchs are automatically generated from all the precompiled header files in [input_models/compiled](#), but these are not emitted from any `.onnx` file: the user has to manually compile any desired ONNX file and put it into the aforementioned directory. For this reason the workflow is semi-automated.

The advantage of commissioning the user to emit the headers is that no direct dependencies with SOFIE are brought into rootbench.

As a recapitulatory plot, I propose in Figure 8 a performance comparison between the state-of-the-art ONNX Runtime and SOFIE. The plot is also comprehensive of the models compiled with the RModelProfiler with the aim of unveiling how low is the overhead introduced by the code instrumentation.

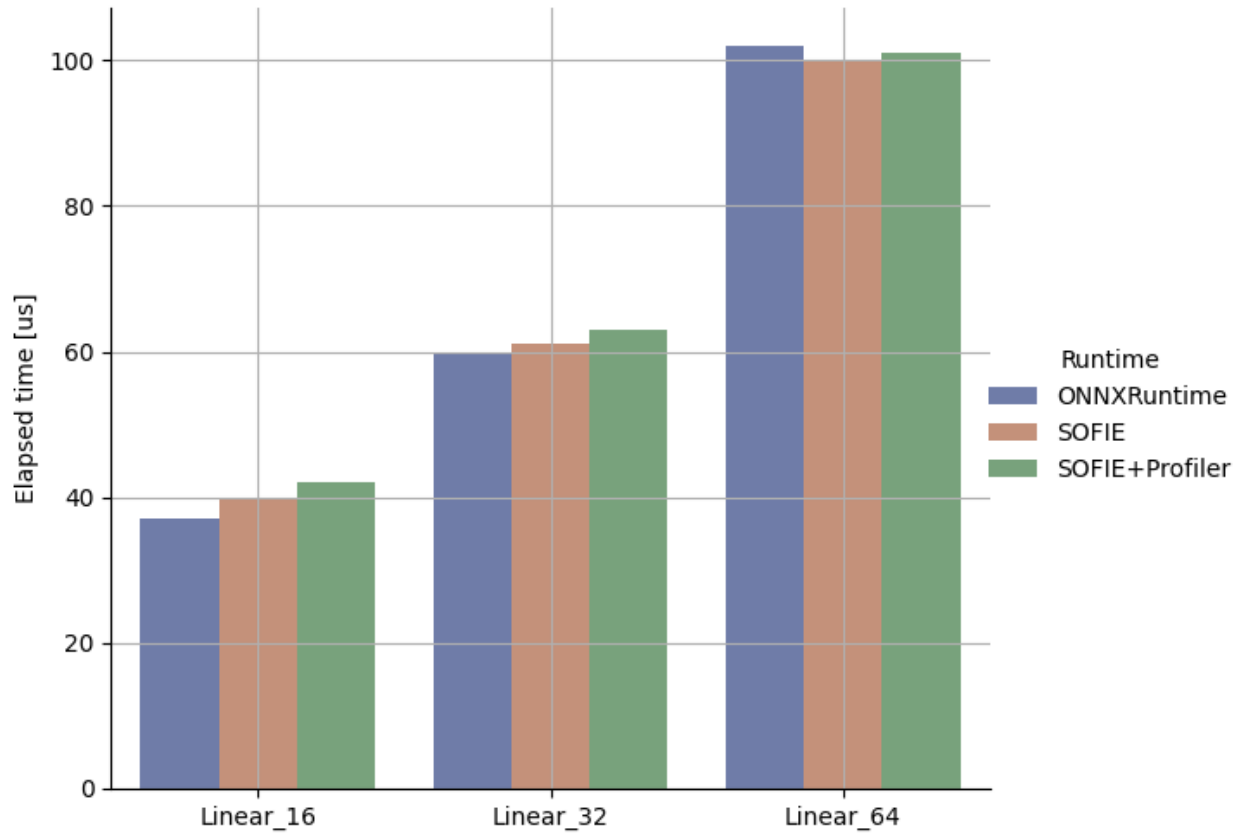


Fig.8: Performance comparison of ONNX Runtime vs SOIFE (+profiled version). OpenBLAS, single thread.

The benchmark is conducted on 3 multi-layer perceptrons composed of 10 layers with ReLU activations in between. The linear algebra library used is OpenBLAS in single thread mode, which gave the best results among other configurations. We observe that SOFIE is competitive with ONNX Runtime and that the profiled version does not introduce much overhead.

Note: Setting the number of OpenMP threads to any number more than 1 will not benefit performance, since for the matrix sizes involved in these benchmarks

CONCLUSIONS

This report summarizes the work that I did on the software project ROOT as a Summer Student at CERN during summer 2021. I briefly described how I implemented GoogleTests and GoogleBenchmarks for the new experimental module of TMVA called SOFIE. I also devised the simple class `RModelProfiler` with the aim of measuring the time that each operator of a given model took during any inference process.

PULL REQUESTS

1) End-to-end GoogleTests:

<https://github.com/root-project/root/pull/8782>

2) RModelProfiler class:

<https://github.com/root-project/root/pull/8957>

3) SOFIE vs ONNX Runtime benchmarks:

<https://github.com/root-project/rootbench/pull/236>

At the time of writing, only PR #8782 has been merged into the master branch, while the others still require some refinements.

ACKNOWLEDGEMENTS

Back in January 2021, when I was preparing my application for this studentship, I never imagined that a few months later I would have received a *Congratulations* email from CERN. Now the Summer Student Programme 2021 came to the end I can proudly say that I have been part of it. I am grateful to CERN, as an organization, for giving me the opportunity to grow, jump in the open source world and meet new people, such as my fellows Sanijban Sengupta, Ahmat Hamdan and Simone Azeglio. Thanks also to Omar Andres Zapata Mesa and the whole ROOT developers team.

But my biggest thanks go to the mentors of this project, Lorenzo Moneta and Sitong An. Thank you for being so nice and friendly to me and for the time that both of you dedicated to guiding me through the complexities of the work and to providing constructive feedback to my ideas. This project really gave me a glimpse on how it feels to be at the forefront of something new and experimental and how huge and important the role of open source is in advancing science.

ABOUT ME

I'm from Italy. I studied Computer Engineering at the University of Padova, IT, where I completed my Master degree in July 2021. I'm passionate about programming and High Performance Computing and now I'm looking for opportunities to further specialize my knowledge.

Feel free to reach me on LinkedIn or by email.



[linkedin.com/in/federico-sossai](https://www.linkedin.com/in/federico-sossai)



federico.sossai@gmail.com



github.com/fsossai
