ONNX Interpreter

Perlo Giacomo 317981
Sanino Fabrizio 317541

academic year 2022/23

Summary

Overview

Onnx Parser

Onnx Serializer

Inference

Multi-Threading

Binding

Results

Overview

ONNX is an open format built to represent machine learning models.

ONNX defines a common set of operators and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

The objective of the project is to build a ONNX Interpreter using the **Rust** language.
The project, indeed, provides:

- a ONNX Model Parser and Serializer
- the opportunity to make Inference on the parsed model by leveraging Multi-Threading paradigm
- the binding towards Python of the Inference functionality

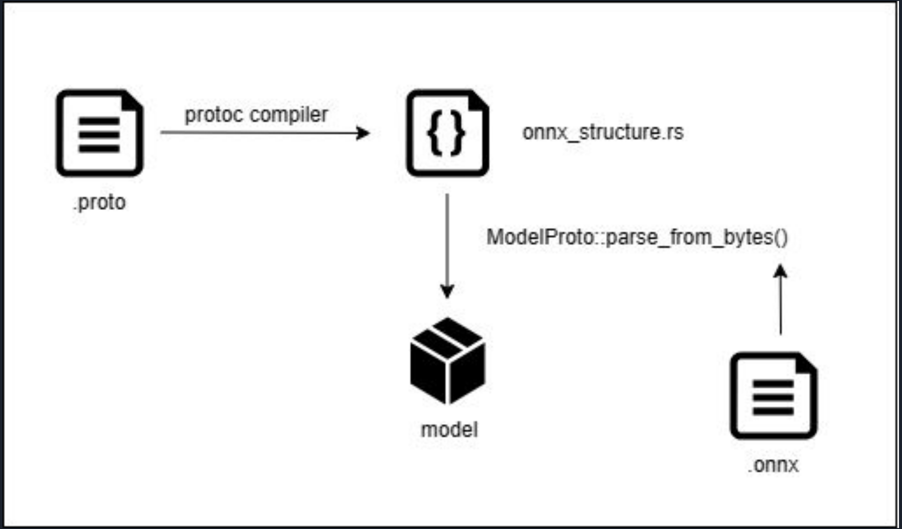
Onnx Parser

The objective of the Parser is to obtain the runtime Onnx model starting from its .onnx file.
The parser work could be split in two main phases:

- Generation of the classes based on the *messages* present in the ***.proto*** file. This operation could be performed with the *protoc* compiler:

  protoc --rust_out=output_dir file.proto

- Each protocol buffer class has methods for **parsing** and **serializing** messages. In our specific case we exploited the ***::parse_from_bytes(****.onnxfile****)*** method which returns the desired runtime onnx structure.

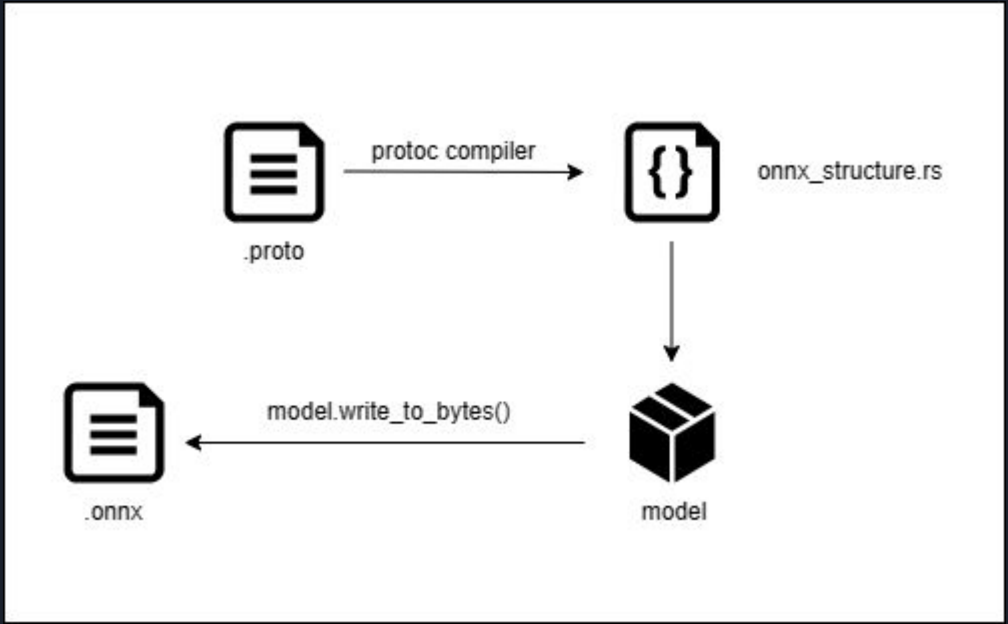


Onnx Serializer

The objective of the Serializer is to write a new .onnx file starting from a runtime onnx model previously read.

Furthermore, this module allows to modify the model such as creating new nodes or changing other features.

The goal is reached by exploiting the ***write_to_bytes()*** method which, once got the runtime model, writes it back on a file accordingly to the encoding of the protocol buffer



Inference

The objective of the **inference** is to execute the *.onnx* model's operation on the input data.

Based on the ***op_type*** of the node, we can distinguish which operation must be performed.

The most significant **operations** we have developed (based on the models that we decided to execute) are:

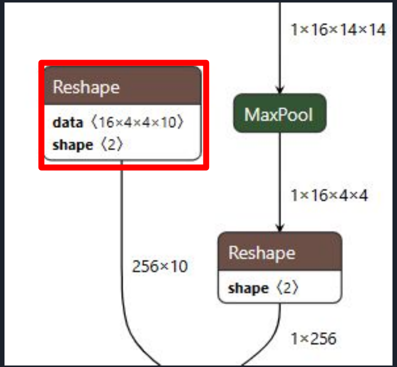
- Convolution: for a given input (2d image) it allows to obtain different convolved features based on the filter applied on the input
- MaxPool: for a given input (2d image) it allows to calculate the maximum value for patches of the image and uses it to create a downsampled map
- Relu: Activation function

Other operations developed: Dropout, GlobalAvergaPool, Reshape, Softmax.

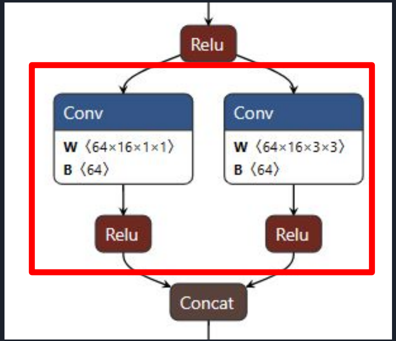
Multi Threading

The objective of exploiting **Multi Threading** is to speed up the Inference process. The two scenarios in which this is possible are:

- **Independent nodes** nodes whose data is available from the beginning of the inference process (since stored in the Model *Initializers)*.



- **Parallel nodes** nodes who share the same input data.



Every time a node (or a group of nodes) respect one of the aforementioned characteristics, a thread (or a group of threads) is spawned. Whenever the following process doesn't have the data needed to execute the inference, then it is suspended on a **condition variable**.

Binding

The objective of the Binding module is to give the possibility of using and exporting functions written in a language into another programming language. It was decided to export Rust functions to **Python**.

The technologies used are the following:

- **py03**: is a Rust bindings for the Python interpreter. After importing it in our project, functionalities that we had want to be exported were wrapped with a specific methods present in those library.
- **maturin**: tool that is used for build Python package.

After having builded the Python .whl file, it's possible to install with pip and the import the library into a Python project.

Results

```
INFERENCE ON INPUT(s) ["Input3", "Parameter5"] OVER Conv OPERATION done by Thread1
Reshape, done! by Thread0
MAIN THREAD WAITING FOR CHILDREN THREADS RESULTS
Convolve, done! by Thread1
INFERENCE ON INPUT(s) ["Convolution28_Output_0", "Parameter6"] OVER Add OPERATION done by main
Add, done! by main
INFERENCE ON INPUT(s) ["Plus30_Output_0"] OVER Relu OPERATION done by main
Relu, done! by main
INFERENCE ON INPUT(s) ["ReLU32_Output_0"] OVER MaxPool OPERATION done by main
MaxPool, done! by main
INFERENCE ON INPUT(s) ["Pooling66_Output_0", "Parameter87"] OVER Conv OPERATION done by main
Convolve, done! by main
INFERENCE ON INPUT(s) ["Convolution110_Output_0", "Parameter88"] OVER Add OPERATION done by main
Add, done! by main
INFERENCE ON INPUT(s) ["Plus112_Output_0"] OVER Relu OPERATION done by main
Relu, done! by main
INFERENCE ON INPUT(s) ["ReLU114_Output_0"] OVER MaxPool OPERATION done by main
MaxPool, done! by main
INFERENCE ON INPUT(s) ["Pooling160_Output_0", "Pooling160_Output_0_reshape0_shape"] OVER Reshape OPERATION done by main
Reshape, done! by main
INFERENCE ON INPUT(s) ["Pooling160_Output_0_reshape0", "Parameter193_reshape1"] OVER MatMul OPERATION done by main
MatMul, done! by main
INFERENCE ON INPUT(s) ["Times212_Output_0", "Parameter194"] OVER Add OPERATION done by main
Add, done! by main

MNist-8 Inference results: Class 3-nth predicted.
Actual Data: [[-49.920094, 11.413024, 36.925835, 24.719719, 4.075505, -15.360391, 5.894409, -19.078056, -0.784606, -18.379814]], shape=[1, 10],
Expected Data: [-49.920094, 11.413038, 36.925827, 24.719717, 4.075502, -15.360382, 5.8944097, -19.078064, -0.78461087, -18.379824]
```

MULTI-THREADING

PREDICTED CLASS