MutateNN: Mutation Testing of Image Recognition Models Deployed on Hardware Accelerators

Nikolaos Louloudakis
n.louloudakis@ed.ac.uk
University of Edinburgh

Perry Gibson
perry.gibson@glasgow.ac.uk
University of Glasgow

José Cano
jose.canoreyes@glasgow.ac.uk
University of Glasgow

Ajitha Rajan
arajan@ed.ac.uk
University of Edinburgh

ABSTRACT

The increased utilization of Artificial Intelligence (AI) solutions brings with it inherent risks, such as misclassification and sub-optimal execution time performance, due to errors introduced in their deployment infrastructure because of problematic configuration and software faults. On top of that, AI methods such as Deep Neural Networks (DNNs) are utilized to perform demanding, resource-intensive and even safety-critical tasks, and in order to effectively increase the performance of the DNN models deployed, a variety of Machine Learning (ML) compilers have been developed, allowing compatibility of DNNs with a variety of hardware acceleration devices, such as GPUs and TPUs. Furthermore the correctness of the compilation process should be verified.

In order to allow developers and researchers to explore the robustness of DNN models deployed on different hardware accelerators via ML compilers, in this paper we propose MutateNN, a tool that provides mutation testing and model analysis features in the context of deployment on different hardware accelerators. To demonstrate the capabilities of MutateNN, we focus on the image recognition domain by applying mutation testing to 7 well-established models utilized for image classification. We instruct 21 mutations of 6 different categories (Layer Node Replacement, Arithmetic Node Replacement, Node Input/Output Modifications, Arithmetic Node Replacement, Arithmetic Types Mutations, Kernel Variables & Stores Mutations, and Conditional Statement Operations Mutations), and deploy our mutants on 4 different hardware acceleration devices of varying capabilities. Our results indicate that models are proven robust to changes related to layer modifications and arithmetic operators, while presenting discrepancies of up to 90.3% in mutants related to conditional operators. We also observed unexpectedly severe performance degradation on mutations related to arithmetic types of variables, leading the mutants to produce the same classifications for all dataset inputs.

APPENDIX

MutateNN is a comprehensive suite for compiling, optimizing, executing and analyzing pretrained DNNs under different computational environment settings. In total, the tool supports:

- Downloading and building Deep Neural Networks.
- Generating mutants from a variety of settings for testing purposes, given many parameterization capabilities.
- Generating device code for execution on different hardware acceleration devices supporting different frameworks such as OpenCL and CUDA.
- Executing inference on those devices for a dataset of inputs, following the necessary pre-processing, dependent on each DNN model.
- Performing analysis against all mutation configurations, for all devices, by supporting a variety of pairwise comparison operators, such as top-1 output label comparison, and Kendall's Tau.
- Allowing debug execution and different optimizations applications on DNN models.

The suite utilizes Apache TVM[**?**]. The mutation transformations are programmed on Relay[**?**], TVM's graph Intermediate Representation (IR), and TIR, the Tensor-level IR.

1 INSTALLATION

The system requires TVM to be installed. We also use `Python v3.x.x` (tested with `3.6.x-3.10.x`) and `Pip` as the package installer.

In addition, the system requires a number of pip packages. You can find them in the `requirements.txt` file

2 INSTRUCTIONS

(1) Install Python and Pip on your system. Python comes with linux distributions usually, but this is not always the case for Pip. You can install it by running `sudo apt install python3-pip`
(2) Download and install TVM: For instructions of how to install TVM, please refer to the TVM related guide for developers[1]. We tested MutateNN using *TVM v0.13.0*.
(3) Follow the installation from source instructions, and based on the experiments you want to run, enable the respective flags in the `<tvm_folder>/build/config.cmake`. For our experiments, we followed different settings per-device, but consider enabling the `USE_LLVM` and `USE_OPENCL` or `USE_CUDA` flags, depending on your configuration.
(4) Install necessary Python packages by executing the command: `pip install -r requirements.txt`
(5) Download necessary models, if you wish to run them locally. Alternative, you can instruct the MutateNN to download them for you. Although system utilizes already provided models for Keras and PyTorch, we utilized some TF and TFlite models from the GitHub repository of TensorFlow for slim Models.
(6) You can also download the models manually, place them in the models folder defined in `models_out_relative` parameter, and defining `"type": "local"` in the configuration file. By default, use models from the official TensorFlow repo[2]. The supported format for the models is `ONNX`.

[1] https://tvm.apache.org/docs/install/from_source.html#developers-get-source-from-github
[2] https://github.com/tensorflow/models/tree/master/research/slim

3 CONFIGURATION

The configuration of the system is included into the 'config.json' file. Each section is self-explanatory and defines which part it concerns.

Important notes:

- You can utilize the TVM debugger, by setting `debug_enabled: true` to collect additional inference metadata and execution traces.
- You can set different TVM optimization settings, by modifying the `opt_level` variable from 0 to 4, with optimization level increasing.
- You can set the mutations you want to generate, by modifying the `mutations` entry of the object.
- You can specify the occurrence numbers that you want your mutation to be applied, by modifying 'positions' parameter in mutations. You can see examples of it on the configuration file provided.
- Device settings have been cleared out to preserve anonymity. If you wish, you can set up your own TVM RPC server on your own device and run everything following the instructions in TVM documentation[3]. You can then define your device in the system configuration and perform inference.

4 EXAMPLE CASE

In order to verify your installation and be able to run the framework with your own configuration, we have setup the configuration to build the system utilizing 7 models under test: MobileNetV2 [**?**], DenseNet121 [**?**], ResNet152 [**? ?**], AlexNet [**?**], EfficientNetLite [**?**], ShuffleNet [**?**], and InceptionV2 [**?**]. You can download, run and evaluate the models accordingly. All models are obtained from the slim official repository, are pre-trained against ImageNet [**?**] and perform classification tasks against 1000 labels.

We also provide a small test dataset, consisting of 5 public domain images, obtained from unsplash[4]. To demonstrate device comparison, we have generated 3 simulations on different devices for `MobileNetV2`, which can be found on `/generated/MobileNetV2/simple_run`. You can instruct `MutateNN` to build, run and evaluate the existing dataset against these device outputs, by setting `build`, `execute` and `evaluate` to `true` in the`MobileNetV2` model entry of the configuration file.

Each model configuration entry also contains a number of necessary parameters, such as the input layer name and size, the output layer, etc, which are necessary for the model preparation, deployment, execution and evaluation.

Once you set up the tool, you can execute `MutateNN` by running: `python3 main.py`. An example of an execution instance terminal output, containing model build, execution and analysis, is presented on figure1.

4.1 Model Build & Mutants Generation

Inside `config.json`, you can set the mutations you want to generate, by modifying the `mutations` entry of the object. You can instruct `MutateNN` to generate mutants on Relay IR, or in the Tensor-level IR. A number of supported mutations are already provided, but they can be modified and parameterized, based on the user needs.

The system will generate the models in the folder defined in `config.json` in a `tar` package, along with a folder providing their generated Relay, TIR representations, but also their GPU host and kernel code, for inspection and debugging purposes.

In total, the framework will generate the models compiled on TVM, utilizing the `opt=2` optimization setting by default which performs basic graph-level optimizations to the models, such as inference simplification, operator fusion and constant folding.

4.2 Execution

Your system will then execute, generating a folder with experiments. The structure followed is the following, using `MutateNN` folder (`<script_folder>`) as the base folder:

- Build: `/<models_out_relative>/<model_variant>_<opt_setting>.tar`
- Execution: `<exec_out_relative>/mutations/ts_<epoch_time_of_run>/<predictions>.txt`
- Evaluation: `<evaluation_out_relative>`

Based on existing configuration, inference generates the top-5 predictions, along with the execution time per-prediction at the bottom. In addition, you will find an `execution_log.txt` file in the aforementioned folder, containing info about the run.

The console will indicate the status of the running model and update accordingly, as shown in Figure 1.

4.3 Analysis

Once execution is complete, analysis will be executed. This will be done in 3 ways:

- Comparing results per-device (if provided), in JSON files.
- Analyze CSV and JSON files, containing metadata related to the execution.
- Comparing results per-multiple executions (if provided).

The system will then generate the following files inside each evaluation folder:

- `device_evaluation.json`, containing results per-device comparison in a pairwise manner.
- `device_discrepancies.json` containing only the cases where dissimilarities are observed.
- `output_stats_total.csv`, containing inference time data and performance of statistical analysis against execution times (using One Way ANOVA). This is an implementation related to analyzing inference times, which is not included in this work but is intended for future work usage.

Finally, you can also try your own model, given you provide the right files and settings. Configuration provides exactly the details requested for a model to be loaded from a backend, compiled using a specific optimization and GPU backend and be run for inference, respectfully.

4.4 Error Logging

In case of an error, the suite will generate a file related to the specific execution instance, by generating a file containing all the necessary

[3] `https://tvm.apache.org/docs/tutorial/cross_compilation_and_rpc.html`
[4] `https://unsplash.com/images/stock/public-domain`

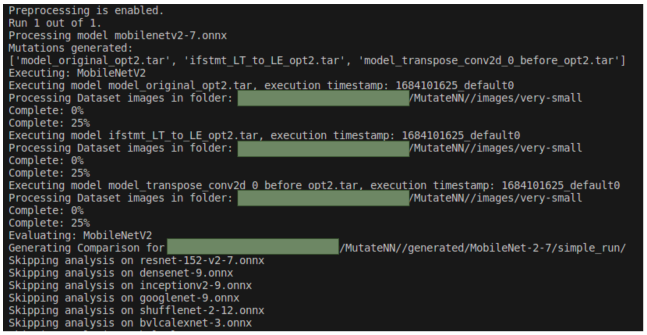

Figure 1: Log of `MutateNN` performing all operations (build, execute, evaluate) simultaneously, for `MobileNetV2`.

data in `<script_folder>/error_log/<model>/ts_<epoch_time_of_problematic_run>/error_log.txt`.

4.5 Tool Demonstration

A video demonstrating `MutateNN` is uploaded and can be found at: https://www.youtube.com/watch?v=j7Ffd5y_i3g.