Neural Network Model Transformation Framework for On-Devices

Kyung Hee Lee
AI Computing System *SW Research Section*
ETRI
Daejeon, Korea
kyunghee@etri.re.kr

Jaebok Park
AI Computing System *SW Research Section*
ETRI
Daejeon, Korea
parkjb@etri.re.kr

Seon-tae Kim
AI Computing System *SW Research Section*
ETRI
Daejeon, Korea
stkim10@etri.re.kr

Ji Young Kwak
AI Computing System *SW Research Section*
ETRI
Daejeon, Korea
jiyoung@etri.re.kr

Hong Soog Kim
AI Computing System SW Research Section
ETRI
Daejeon, Korea
kimkk@etri.re.kr

Chang Sik Cho
AI Computing System *SW Research Section*
ETRI
Daejeon, Korea
cscho@etri.re.kr

***Abstract*— On-device systems generally have limitations on computational resources, and there are various types of hardware and neural network engines supporting them. Due to the limitations of computational resources, on-device systems often only execute inference using a model trained in a personal computer or server system. Therefore, in order to use the trained model in various on-device environments, it is necessary to provide functions that support transformation of the neural network model, optimization of the neural network model, and execution of inference for each neural network engine to suit the on-device environment using the neural network model. In this paper, we implemented an on-device neural network model transformation framework that receives the trained neural network model and the environment information of the target on-device and creates a neural network model that can be performed on-device. This framework improved transformation efficiency by using a standard neural network model format based on an open source project called ONNX, and supported neural network inference engines such as TensorRT, TVM, and PyTorch. In the future, this framework will be supplemented with functions such as automatic generation of neural networks, linkage with neural network learning frameworks, and expansion of support for various inference engines.**

Keywords—Neural Network, Model Transformation, ONNX

I. Introduction

Neural networks are used in various fields such as image recognition, voice recognition, and language recognition. In order to implement such an artificial intelligence service using a neural network, a neural network structure is defined, and a neural network model is trained by using a neural network engine and data for learning. Then, in order to obtain an inference result such as image recognition, a neural network execution engine is installed on the hardware where the neural network is to be executed, and data to be recognized is input to the trained neural network model to obtain a result.

As neural network engines that train or infer neural networks, PyTorch[1], Tensorflow[2], and Caffe[3] have been developed. Neural network training requires a large amount of data processing and high-performance computational performance. On the other hand, the inference process can be performed even in an environment where computational resources are relatively scarce. Therefore, most of the above engines perform the learning process on a personal computer or a server computer. However, the inferencing process may be executed on a personal computer or a server computer depending on the type of service, or may also be executed on an on-device such as an embedded system.

This paper deals with the implementation of a framework that transforms neural network models trained in other systems and generates execution codes for on-device so that they can be executed on-devices.

This neural network transformation framework receives the neural network model trained in PyTorch and user requirements, and provides functions to automatically generate executable codes on the target on-device.

This paper describes the characteristics of on-device in Chapter 2, deals with the standard model format for neural network in Chapter 3, discusses the structure and operation process of the framework in Chapter 4, and describes the conclusion in Chapter 5.

II. Features of On-Devices

On-device refers to an embedded system with limited memory size and computational performance. Unlike personal computers or server computers, on-device has limitations in the performance of hardware such as a central processing unit, graphic device, and memory. In addition, CPU (Central Processing Unit), GPU (Graphic Processing Unit), and NPU (Neural Processing Unit) used in on-device are more diverse than those of personal computers or servers.

Depending on the diversity of on-device hardware, the types of operating system used are also diverse, such as embedded Linux, Android, and Windows.

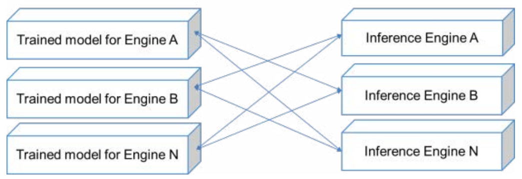
In addition, various engines for neural network inference are provided by GPU manufacturers, NPU manufacturers, or device manufacturers.

Therefore, in order to execute a trained neural network model on a personal computer or server system, the trained model must first be transformed into a model for the inference engine supported by the on-device, and optimization of the neural network model to suit the on-device hardware environment must be supported. For developers who are not familiar with the on-devices to cope with the diversity of the on-device, sample execution code generation will be helpful for them.

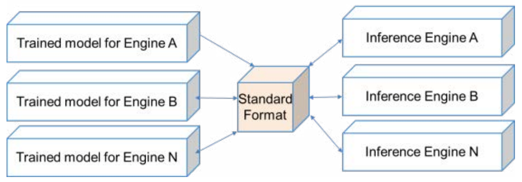
III. Standard Formats for Neural Network Models

The neural network model is a data structure that stores the weights in the neural network graph structure and the neural network nodes. The neural network model format depends on the neural network engine used. Each neural network engine defines its own neural network model structure.

Therefore, the developer must transforms the neural network model when the engines used for training and inference are different. At this time, in the case of N:N transformation for each learning engine and reasoning engine, the structure becomes complicated and the efficiency will be decreased as shown in (a) in the figure below. In order to overcome this problem, a structure that defines a standard format expressing a neural network model and transforms it based on this can increase efficiency in the implementation and reusability of the transformer.


Trained model for Engine A
Trained model for Engine B
Trained model for Engine N
Inference Engine A
Inference Engine B
Inference Engine N

(a) N:N Transformation


Trained model for Engine A
Trained model for Engine B
Trained model for Engine N
Standard Format
Inference Engine A
Inference Engine B
Inference Engine N

(b) Neural Network Model Transformation based on Standard Format

Fig. 1. The Types of Neural Network Model Transformation

Typical neural network standard formats are ONNX(Open Neural Network eXchange)[4] and NNEF(Neural Network Exchange Format)[5]. ONNX is a format released as an open source, and NNEF is a format established by a standardization organization called Khronos Group.

NNEF supports dynamic neural network definition by describing neural network model in a language similar to Python. ONNX stores the neural network structure and trained data with the protocol called Protobuf[6], and it saves graph information and learning results in a single file. Since this is an open source project rather than a standardization organization, adding and improving its functions is relatively fast. In addition, ONNX has the advantage of being developed even before the release of NNEF, the source is open, and there are many users.

Since NNEF goes through the ratification process in its hierarchical organizational, it takes a lot of time to improve standards, and the number of neural network engines supported is very small compared with ONNX. Therefore, in this paper, ONNX, which supports a large number of engines, was adopted as a standard format.

IV. Neural Network Model Transformation Framework

The neural network model transformation framework receives a model trained with PyTorch, a trained weight file, and transformation information. This transformation information contains information such as hardware and software environments of the target on-device and requirements for optimizing the neural network model.

The neural network input model used PyTorch, which was used frequently in artificial intelligence industry, and the transformation information used YAML (Yet Another Markup Language) format.

Target engines supported by this framework are PyTorch, TensorRT provided by NVIDIA, and TVM of Apache Foundation. This framework basically uses PyTorch as an input model, and allows ONNX to be input as well. It adopts ONNX as an intermediate format for ease of transformation between the trained model and the model of the target inference engine.

For neutrality of operating systems such as Linux and Windows, this framework sets the Python language as the base programming language to generate neural network codes.

TABLE I. Main Functions of the Framework

Category	Supported Items
Inference Engine	PyTorch, TensorRT, TVM
OS	Linux, Windows
Input Neural Network Model	PyTorch, ONNX
Programming Language	Python

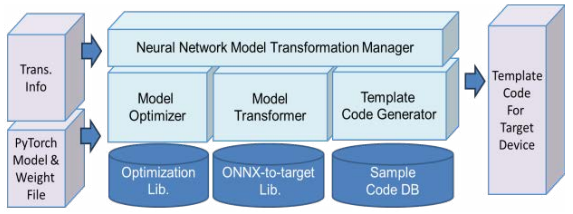


Fig. 2. The structure of Neural Network Model Transformation Framework

A. *The Structure of the Framework*

The neural network model transformation framework has the structure shown in Fig 2. What is expressed on the left side of this figure are the transformation information containing the on-device environment and user requirements, which are expressed in YAML format.

In addition, this framework additionally receives a neural network model trained with PyTorch and a weight data file. The right part of the figure is the output result, which includes the neural network model transformed and optimized for the target on-device environments and the template codes to run this model on the target.

Neural Network Transformation Manager provides an API to receive neural network model and transformation information from the user and controls the operation pipeline of modules within the framework.

Model Optimizer is to optimize the neural network model. It transforms the input PyTorch neural network model into ONNX and quantizes the weight data according to the user's optimization request, thereby reducing the size of the weight data and improving the execution speed. This module basically uses the lightweight function provided by ONNX. Optimization Lib. includes ONNX's library that executes various algorithms for optimizing neural networks.

Model Transformer provides a function to transform the ONNX neural network model to the target engine's neural network model format. ONNX-to-target Lib. includes a transformation library that transforms ONNX neural network models into formats used by target inference engines.

The Template Code Generator automatically generates codes of the pre-processing for input image and the post-processing for output data after neural network inference, so that the original neural network model can be executed on the target on-device.

The pre-processing for inputting the neural network includes functions of transforming the format and size of the input image to fit the neural network. The post-processing includes functions of analyzing the output of the neural network to extract ID of the recognized object, the location of the object, and accuracy information. The Sample Code DataBase includes example codes for executing neural networks for each inference engine, a library of pre-processing codes for neural network input, and a library of post-processing codes for neural network output.

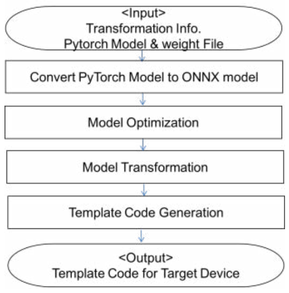


Fig. 3. Operation Flows in The Framework

B. *Functional Tests of the Framework*

To test the functions of this framework, a YAML file for transformation information was created, and a neural network model trained with PyTorch and a weight data file were generated. In this test, YoloV5 was used for a neural network, and the neural network was trained using the COCO (Common Objects in Context) data sets. By inputting these into this framework, the output shown in the figure below was obtained, and the operation was confirmed by executing it on the Jetson Nano device.

```
# General Info.
task_type : etection
target_info : ondevice
cpu : x86
acc : cpu
memory: 32
os : ubuntu
engine : tensorrt
# NN Model Info.
class_file: ['yoloe.py']
class_name: 'Model()'
weight_file: yoloe.pt
# Input
input_tensor_shape: [1, 3, 640, 640]
input_data_type: fp32
# Output
output_number: 3
output_size:  [[1, 3, 20, 20, 85], [1, 3, 40, 40, 85], [1, 3, 80, 80, 85]]
# Post-processing
conf_thres: 0.25           # for NMS
iou_thres: 0.45           # for NMS
```

Fig. 4. Transformation Information File

```
import tensorrt as trt
import pycuda.autoinit
import pycuda.driver as cuda
class TRTRun():
  def load_model(self):
    TRT_LOGGER = trt.Logger(trt.Logger.WARNING)
    runtime = trt.Runtime(TRT_LOGGER)
    trt.init_libnvinfer_plugins(None, "")
    with open(self.model_path, 'rb') as f:
      self.engine = runtime.deserialize_cuda_engine(f.read())
    self.context = self.engine.create_execution_context()
    for i in range(self.engine.num_bindings):
      is_input = False
      name = self.engine.get_tensor_name(i)
      isBN = self.engine.get_tensor_mode(name)
      if isBN == trt.TensorIOMode.INPUT:
        is_input = True
       tmptype = trttype2nptype(self.engine.get_tensor_dtype(name))
      dtype = np.dtype(tmptype)
      shape = self.context.get_tensor_shape(name)
      if is_input and shape[0] < 0:
        profile_shape = self.engine.get_profile_shape(0, name)
        self.context.set_binding_shape(i, profile_shape[2])
        shape = self.context.get_binding_shape(i)
...
```

Fig. 5. Parts of Generated Neural Network Execution Codes for TensorRT

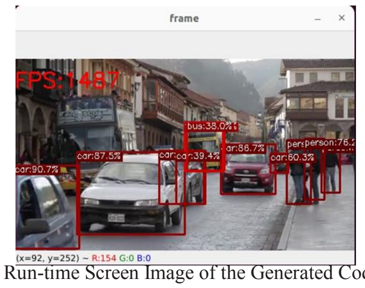


Fig. 6. Run-time Screen Image of the Generated Codes

V. Concluding Remarks

In this paper, the implementation of a neural network model transformation framework for on-device is described. This framework receives information about models trained with PyTorch, training data, and target on-device. This framework transforms and optimizes the neural network model so that the neural network model can operate on the target with input received from the user, and generates executable codes that can operate on the target engine and operating system.

In the future, this framework will have deployment functions for the other on-devices, cloud systems, etc. In addition, it will be expanded to a neural network development tool through linkage with the automatic neural network generation framework or neural network training framework.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2021-0-00766, Development of Integrated Development Framework that supports Automatic Neural Network Generation and Deployment optimized for Runtime Environment)

References

[1] https://www.pytorch.org
[2] https://www.tensorflow.org
[3] https://caffe.berkeleyvision.org
[4] https://onnx.ai
[5] Khronos Group, Neural Network Exchange Format, Version 1.0, Khronos Group, October 2017.
[6] https://protocolbuf.dev