TINYML in TmallGenie
1. Motivation & Backends
2. TinyML Framework: Pruning, Quantize and others
3. HA NAS: TinyML for NPUs
4. Future Works
The number of AI models deployed on Tmall Genie and other AIoT devices is growing rapidly.
Common problems to be tackled in PyTorch-based TinyML framework

1. The dynamic graph used by PyTorch makes operator dependency analysis difficult, which poses challenges for automatic pruning and quantization of complex models.
2. PyTorch models do not translate well to TFLite models, making them difficult to deploy on many AIoT devices.
3. When the pruning rate is high (for example, more than 95% of the flop is compressed), the effectiveness of existing pruning methods will be severely deteriorated.
4. Other works only generate tiny models, but still have a lot of works to do to run the model in devices, this is hard to use, we need a end to end framework to make TinyML more easier to use.
The End to End TinyML Framework
Architecture

Quantized Model
- Quantization Aware Training
- Insert & Fuse QAT OP

Pruned Model
- Prune
- Operators Dependence Analysis
- Dynamic Graph Tracer
- Compute Graph
- AI Model & Code

Model & Code Generator

Quantized Model

Hardware Information

TFLite Converter

AI Model & Code

AloT Devices
**Graph Tracer**

1. **Pre-call hook**
2. **forward()**
3. **post-call hook**

**torch.nn.Module**

**Function wrapper**

- `torch.foo()`
- `torch.Tensor.foo()`
- `torch.functional.foo()`
- `torch.nn.functional.foo()`

**Application**

1. Construct the compute graph from the model via tracing.
2. Generate the corresponding model definition code from the compute graph.
Dependency analysis
1. Divide the computational graph into multiple independent subgraphs.
2. Resolve channel dependency in the subgraphs.

Pruning
1. Use L1, L2, FPGM, ADMM, NetAdapt and other algorithms to prune the neural networks.
2. We propose a new progressive pruning algorithm, which reduces the accuracy deterioration under high pruning rate. (ICLR Reviewing)

Generating code
1. Generate the corresponding model definition code so that the pruned weights could be loaded.
Quantization-aware Training

Graph rewriting
1. Insert FakeQuant nodes.
2. Replace the nodes that cannot be quantized with their quantizable equivalents.

Code generation
1. Generate the corresponding model definition code according to the rewritten compute graph.
2. The FakeQuant nodes may be removed or added in the generated model definition code to achieve mixed precision. (By default, we try to quantize as much as we can in the entire model).

QAT Preparation & Training
1. Operator fusion and preparation for quantization-aware training.
The TFLite Converter is a tool for converting models from various frameworks to TensorFlow Lite. The current pipeline has several limitations:

1. Lengthy procedure leads to insufficient stability
2. Lack of support for quantized models
3. The converted model has a lot of redundant operators, thus leads to slow inference

Our converter addresses these issues:

1. End-to-end conversion
2. Support quantized models
3. Support graph optimization (Transpose elimination, Slice fusion, BN and Activation fusion, etc.)

<table>
<thead>
<tr>
<th>Model</th>
<th>ONNX-&gt;TF-&gt;TFLite</th>
<th>TinyML Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2</td>
<td>810 ms</td>
<td>384 ms</td>
</tr>
<tr>
<td>MobileNetV3</td>
<td>196 ms</td>
<td>106 ms</td>
</tr>
</tbody>
</table>

The experiment is performed on a Tmall Genie with MTK 8167.
model = mobilenet.Mobilenet()
model.load_state_dict(torch.load(WEIGHT_PATH))

dummy_input = torch.ones(1, 3, 224, 224)

pruner = OneShotChannelPruner(model, dummy_input, config)
pruner.prune()

# Model finetune (custom code)
finetune(model)

quantizer = QATQuantizer(model, dummy_input, work_dir="./out")
qat_model = quantizer.quantize()

# Quantization aware training (custom code)
qat_finetune(qat_model)

converter = TFLiteConverter(qat_model, dummy_input, tflite_path="./out/mobv1_q.tflite")
converter.convert()
Got an average speedup of 5.3x on 16 business models for Tmall Genie.
HA NAS: TinyML for NPUS
Challenge: Designing efficient & accurate models for NPUs
Solution: Hardware–Aware NAS

- Backbone
- Supernet
- Candidates
- Hardware Constraints
- Operator Space
- Search Algorithm
- Evaluation

Deploy

Model

HA NAS

Alibaba
• RT measured on MTK 8175, evaluating validation accuracy using cifar-10
• Baseline: Model with 75% sparsity using one-shot L2 pruning
Future Works

Hardware aware NAS.
More automation makes for better results.
Develop new algorithms for better pruning, quantization and etc.
TF support and other features that made the framework easier to use.
We have made our framework open source in Github: alibaba/TinyNeuralNetwork
Look forward to see your contribution!
Let’s make ML tinier together!

THANKS