tinyML Summit

Miniature dreams can come true...

March 28-30, 2022 | San Francisco Bay Area

www.tinyML.org
Speech models design and deployment on device.

Oleg Rybakov
rybakov@google.com
Catalyst team
Agenda:

- **Performance (latency, memory footprint, power consumption):**
  - Streaming
    - Functional API
    - Subclass API
  - Quantization
    - Post training quantization
    - Quantization aware training
      - Fake
      - Native

- **Applications:**
  - Hotword detection:
    - Multihead self attention
    - Matchbox
  - Streaming multihead self attention
Model streaming (using ring buffers: State1, State2):

Input sample

Previous State1 \quad \text{t-1} \quad \text{State1} \quad \text{t} \quad \text{Next State1}

Previous State2 \quad \text{t-1} \quad \text{State2} \quad \text{t} \quad \text{Next State2}

Convolution 3x3

Dense

RingBuffer
Functional API:

Non streaming model:
net = tf.keras.layers.Conv2D(...)(net)
net = tf.keras.layers.Flatten(...)(net)
net = tf.keras.layers.Dense(...)(net)

Streaming aware model:
net = Stream(cell=tf.keras.layers.Conv2D(...))(net)
net = Stream(cell=tf.keras.layers.Flatten(...))(net)
net = tf.keras.layers.Dense(...)(net)

Applications:
- Real-time Speech Frequency Bandwidth Extension. Yunpeng Li et al
- Streaming keyword spotting on mobile devices. Oleg Rybakov, et al
- Real time spectrogram inversion on mobile phone. Oleg Rybakov, et al
Streaming aware model
(trained in non streaming mode)

Input [1, 16000]
Stream(cell=tf.keras.layers.Conv2D(...))
Stream(cell=tf.keras.layers.Flatten(...))
  tf.keras.layers.Dense(...)  

Streaming inference

Input [1, 200]
RingBuffer
  tf.keras.layers.Conv2D(...)  
  RingBuffer  
  tf.keras.layers.Flatten(...)  
  tf.keras.layers.Dense(...)  

Subclass API, **lingvo**, **keras**

```python
def __init__(self, label_count, apply_quantization, **kwargs):

    self.conv = ring_buffer.RingBuffer(quantize.quantize_layer(tf.keras.layers.Conv2D()))  # Create quantization, streaming aware layer

def call(self, inputs):  # Forward propagation for model training

    net = self.conv(inputs)

    return net

def stream_inference(self, inputs, states):  # Inference in streaming mode

    net, output_state = self.conv(net, state=states)

    return net, output_state
```
Edge cases: causal vs non causal conv
Why model quantization?

- Speed up computation.
- Reduce memory footprint.
- Reduce power consumption.
Post training quantization (PTQ) with TFLite

● Pros:
  ○ Dynamic quantization easy to use, works on most models.
  ○ With int8 quantization can give speed up: 1.5x…4x e.g. on hotword models.
  ○ Supports full model quantization, but needs representative data sets.

● Cons:
  ○ Does not support lower bits e.g. int4.
  ○ There is numerical difference between float and quantized models.
  ○ On more complicated model e.g. auto-regressive models numerical difference can increase.
Fake quantization aware training (QAT)

- **Pros:**
  - Supports lower than 8bits quantization (inference engine also has to support it).
  - Does not need calibration data for full model quantization (vs post training quantization).
  - Easy to use on functional tf.keras.

- **Cons:**
  - During model training it uses fake quantization (uses float ops), if results of float summation and multiplication does not fit into 23 bits mantissa there will be a numerical difference between forward propagation in training and inference modes (when integer runs with int ops).
  - After model training it needs post training quantization step which will convert float ops to int ops.

\[
\text{matmul} \left( \text{float} \left( \text{int8} (\text{weight} \ast \text{scale}_w) \right), \text{float} \left( \text{int8} (\text{act} \ast \text{scale}_a) \right) \right) \\
\text{scale}_w \ast \text{scale}_a
\]

\[
\text{int8}(x) = \text{np.round}(\text{np.clip}(x, -127, 127))
\]

“Emulates dequantization”
Native quantization aware training (QAT)

- **Pros:**
  - What you train is what you serve: no need in post training quantization step.
  - Speed up not only for inference but for training too (will use int8, int4 ops).
  - Forward propagation during training and inference are numerically the same.

- **Cons:**
  - User will have to manage quantized types and variables (need int8/int4 types), including custom gradients for int ops.

\[
\text{matmul} \left\{ \text{int8(weight * scale_w)}, \text{int8(act * scale_a)} \right\}
\]

\[
\text{scale_w * scale_a}
\]
Observations:

- int8 activation + int8 weights. QAT and PTQ works.
- int8 activation + int4 weights. only QAT works on most models.
- int4 activation + int4 weights. only QAT mostly works on very large models, can require some model tuning.
- < int4 needs significant model modifications. e.g. PokeBNN
Applications: **Hotword detection**

Google Speech commands data V2 with 12 labels:
"yes", "no", "up", "down", "left", "right", "on", "off", "stop", and "go"; "silence" "unknown"

<table>
<thead>
<tr>
<th>Model</th>
<th>Pixel4 CPU[ms] Latency of processing 1 sec in non streaming</th>
<th>Pixel4 CPU[ms] Latency of processing 20ms in streaming mode</th>
<th>Accuracy[%]</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHAtt-RNN (non causal)</td>
<td>13</td>
<td>N/A</td>
<td>98.4</td>
<td>750K</td>
</tr>
<tr>
<td>Matchbox (non causal)</td>
<td>3.0</td>
<td>N/I</td>
<td>98.0</td>
<td>75K</td>
</tr>
<tr>
<td>Matchbox (causal)</td>
<td>3.0</td>
<td>0.2</td>
<td>97.4</td>
<td>75K</td>
</tr>
</tbody>
</table>
Transformer single head self attention (causal)

**Input speech:**

**Local speech feature emb:**

**Keys:**

**Softmax:**

**Values:**

**Query**
Hotword detection

MHAtt-RNN (non causal)
- Conv + biLSTM
  - Multi Head Self attention
  - Self Attention Softmax values

Matchbox (non causal)
- Residual Depthwise conv
  - GlobalAveragePooling2D
  - No attention
Streaming Multihead Self Attention

\[ [B=1, T=1, D=6] \]

- **Value projection**
  \[ [B=1, T=1, N=3, H=2] \]

- **Key projection**
  \[ [B=1, T=1, N=3, H=2] \]

- **Query projection**
  \[ [B=1, T=1, N=3, H=2] \]

- **Ring Buffer**
  \[ [B=1, S=5, N=3, H=2] \]

- **Echo buffer**
  \[ [B=1, S=5, N=3, H=2] \]

- **Einsum**
  \[ B \cdot S \cdot N \cdot H, B \cdot T \cdot N \rightarrow B \cdot T \cdot N \cdot H \]

- **Softmax(in time)**
  \[ [B=1, T=1, N=3, S=5] \]

- **Output projection**
  \[ [B=1, T=1, D=6] \]

- **Posteriors**
  \[ (t) \]

\[ (t-1) \text{ Values} \]
\[ [B=1, S=5, N=3, H=2] \]

\[ (t-1) \text{ Keys} \]
\[ [B=1, S=5, N=3, H=2] \]

\[ (t+1) \text{ Values} \]
\[ [B=1, T=1, N=3, S=5] \]

\[ (t+1) \text{ Keys} \]
Conformer encoder for ASR, S2S.

Processing Audio every 20ms with Speech frontend and Conformer Encoder (17 conformer layers with causal local attention = 60).

<table>
<thead>
<tr>
<th>conformer encoder</th>
<th>Pixel6 CPU[ms] with TFLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>float32</td>
<td>31ms/960MB</td>
</tr>
<tr>
<td>post-quantized (int8)</td>
<td>13ms/160MB</td>
</tr>
</tbody>
</table>
References

- Yunpeng Li et al Real-time Speech Frequency Bandwidth Extension.
- Oleg Rybakov, et al Streaming keyword spotting on mobile devices.
- Oleg Rybakov, et al Real time spectrogram inversion on mobile phone.
- Anmol Gulati et al Conformer: Convolution-augmented Transformer for Speech Recognition.
- Benoit Jacob et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference
- Zhewei Yao et al. HAWQ-V3: Dyadic Neural Network Quantization
- Yichi Zhang et al. PokeBNN: A Binary Pursuit of Lightweight Accuracy
- Hao Wu. LOW PRECISION INFERENC ON GPU
- Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition
- https://www.tensorflow.org/lite
- https://github.com/tensorflow/lingvo/blob/master/lingvo/core/conformer_layer.py
- https://github.com/tensorflow/lingvo/blob/master/lingvo/core/batch_major_attention.py
- https://github.com/google-research/google-research/blob/master/kws_streaming
- https://www.tensorflow.org/model_optimization/guide/quantization/training_comprehensive_guide
Appendix
Design patterns for Stream layer

- **Buffer**: RNN layers: [GRU](#), [LSTM](#)
- **Ring buffer**: Conv1D, Conv2D, DepthwiseConv2D, Flatten, GlobalMaxPooling2D, GlobalAveragePooling2D layers
- **Remainder buffer**: Conv1DTranspose
- ...


tinyML Summit 2022 Sponsors
Copyright Notice

This presentation in this publication was presented as a tinyML® Summit 2022. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyml.org