Machine learning on mobile and edge devices with TensorFlow Lite
Daniel Situnayake
@dansitu

Google
Developer advocate for TensorFlow Lite

Co-wrote this book ➔

TinyML
Machine Learning with TensorFlow Lite on Arduino and Ultra-Low Power Microcontrollers
Pete Warden & Daniel Situnayake
TensorFlow Lite is a production ready, cross-platform framework for deploying ML on mobile devices and embedded systems.
Goals

Inspiration
See what’s possible with machine learning on-device

Understanding
Learn how on-device machine learning works, the things it can do, and how we can use it

Actionable next steps
Know what to learn next, and decide what to build first
What is machine learning?
Data

calcPE(stock) {
    price = readPrice();
    earnings = readEarnings();
    return (price/earnings);
}

Rules
(Expressed in Code)

Answers
(Returned From Code)
if (ball.collide(brick)){
    removeBrick();
    ball.dx=-1*(ball.dx);
    ball.dy=-1*(ball.dy);
}
Traditional Programming

Rules

Data

Answers
Activity recognition

if(speed<4){
    status=WALKING;
}

if(speed<4){
    status=WALKING;
}

else {
    status=RUNNING;
}
Activity recognition

if(speed<4) {
    status=WALKING;
} else if(speed<12) {
    status=RUNNING;
} else {
    status=BIKING;
}
Activity recognition

```java
if(speed<4){
    status=WALKING;
} else if(speed<12){
    status=RUNNING;
} else {
    status=BIKING;
}
```

// Oh crap
Traditional Programming

Rules → Data → Answers

Machine Learning

Answers → Data → Rules
Activity Recognition

- Label = WALKING
  - 0101001010100101010
  - 1001010101001011101
  - 0100101010100101001
  - 010100101010101010

- Label = RUNNING
  - 1010100101001010101
  - 0101010010010010001
  - 0010011111010101111
  - 1010100100111101011

- Label = BIKING
  - 001010011111010101
  - 1101010111010101110
  - 1010101111010101011
  - 11111000111101011

- Label = GOLFING
  - 111111111010011101
  - 001111101011110101
  - 0101110101010101110
  - 1010101010100111110
Demo:
Machine learning in 2 minutes
What inference looks like

1. Load your model
2. Transform data
3. Run inference
4. Use the resulting output
Application code

**Pre-processing**
Transforms input to be compatible with model

**Interpreter**
Runs inference using the model

**Post-processing**
Interprets the model's output and makes decisions

**Model**
Trained to make predictions on data

**Input data**

**Output**
Application code

Pre-processing
Transforms input to be compatible with model

Model
Trained to make predictions on data

Interpreter
Runs inference using the model

Post-processing
Interprets the model's output and makes decisions

Input data

Output

TensorFlow Lite
Understanding TensorFlow Lite

- Introduction
- Getting started with TensorFlow Lite
- Making the most of TensorFlow Lite
- Running TensorFlow Lite on MCUs
Edge ML Explosion
Edge ML Explosion

- Lower latency & close knit interactions
Edge ML Explosion

- Lower latency & close knit interactions
- Network connectivity
Edge ML Explosion

● Lower latency & close knit interactions
● Network connectivity
● Privacy preserving
On device ML allows for a new generation of products
1000’s of production apps use it globally.
Have now deployed TensorFlow Lite in production.

More than 3B+ mobile devices globally.
Android & iOS
Embedded Linux (Raspberry Pi)
Hardware Accelerators (Edge TPU)
Microcontrollers

TensorFlow Lite
Why on-device ML is amazing

What makes it different?
ML on the edge
ML on the edge
Bandwidth
Latency
Privacy & security
Complexity
Challenges

- Uses little compute power
Challenges

- Uses little compute power
- Works on limited memory platforms
Challenges

- Uses little compute power
- Works on limited memory platforms
- Consumes less battery
We’re simplifying on-device ML

Convert once, deploy anywhere
Getting Started with TensorFlow Lite

Model conversion and deployment
Dance Like

Built on TensorFlow Lite using the latest in segmentation, pose and GPU techniques all on-device.
We’ve made it easy to deploy ML on-device
Pick a model
Pick a new model or retrain an existing one.

Convert
Convert a TensorFlow model into a compressed flat buffer with the TensorFlow Lite Converter.

Deploy
Take the compressed .tflite file and load it into a mobile or embedded device.

Optimize
Quantize by converting 32-bit floats to more efficient 8-bit integers or run on GPU.
Workflow

1. Get a TensorFlow Lite model
2. Deploy and run on edge device
Image classification
Identify hundreds of objects, including people, activities, animals, plants, and places.
See model

Object detection
Detect multiple objects with bounding boxes. Yes, dogs and cats too.
See model

Pose estimation
Estimate poses for single or multiple people. Imagine the possibilities, including stick figure dance parties.
See model

Smart reply
Generate reply suggestions to input conversational chat messages.
See model

Segmentation
Pinpoint the shape of objects with strict localization accuracy and semantic labels. Trained with people, places, animals, and more.
See model

Style transfer
Apply any styles on an input image to create a new artistic image.
See model

Text classification
Categorize free text into predefined groups. Potential applications include abusive content moderation, tone detection and more.
See model

Question and answer
Answer user queries based on information extracted from a given text archive.
See model
Image Segmentation

Bokeh effect

Replace background
PoseNet

Estimate locations of body and limbs
MobileBERT

Answer users’s questions on a corpus of text
Text
Classification
Recognition
Text to Speech
Speech to Text

Speech
Text to Speech
Speech to Text

Image
Object detection
Object location
OCR
Gesture recognition
Facial modelling
Segmentation
Clustering
Compression
Super resolution

Audio
Translation
Voice synthesis

Content
Video generation
Text generation
Audio generation
Converting Your Model

TensorFlow (keras or estimator) ➔ SavedModel ➔ TF Lite converter ➔ TF Lite model
# Build and save Keras model.
model = build_your_model()

tf.keras.experimental.export_saved_model(model, saved_model_dir)

# Convert Keras model to TensorFlow Lite model.
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()
Running Your Model

Load your model ➔ Preprocess data ➔ TF Lite interpreter ➔ Use the resulting output
private fun initializeInterpreter() {
    val model = loadModelFile(context.assets)
    this.interpreter = Interpreter(model)
}

private fun classify(bitmap: Bitmap): String {
    val resizedImage = Bitmap.createScaledBitmap(bitmap, ...)
    val inputByteBuffer = convertBitmapToByteBuffer(resizedImage)
    val output = Array(1) { FloatArray(OUTPUT_CLASSES_COUNT) }

    this.interpreter?.run(inputByteBuffer, output)
    ...
}
private fun initializeInterpreter() {
    val model = loadModelFile(context.assets)
    this.interpreter = Interpreter(model)
}

private fun classify(bitmap: Bitmap): String {
    val resizedImage = Bitmap.createScaledBitmap(bitmap, ...)
    val inputByteBuffer = convertBitmapToByteBuffer(resizedImage)
    val output = Array(1) { FloatArray(OUTPUT_CLASSES_COUNT) }

    this.interpreter?.run(inputByteBuffer, output)
    ...
}
private fun initializeInterpreter() {
    val model = loadModelFile(context.assets)
    this.interpreter = Interpreter(model)
}

private fun classify(bitmap: Bitmap): String {
    val resizedImage = Bitmap.createScaledBitmap(bitmap, ...)
    val inputByteBuffer = convertBitmapToByteBuffer(resizedImage)
    val output = Array(1) { FloatArray(OUTPUT_CLASSES_COUNT) }

    this.interpreter?.run(inputByteBuffer, output)
    ...
}
The new **TF Lite Support Library** makes development easier

- APIs for simplifying pre- and post-processing (launched)
- Autogenerates pre- and post-processing (in progress)
/** Without TensorFlow Lite Support Library */

ImageClassifier(Activity activity) throws IOException {
    tfliteModel = loadModelFile(activity);
    tflite = new Interpreter(tfliteModel, tfliteOptions);
    imgData = ByteBuffer.allocateDirect(
        Dim.BATCH_SIZE
        * getImageSizeX()
        * getImageSizeY()
        * DIM_PIXEL_SIZE
        * getNumBytesPerChannel());
    imgData.order(ByteOrder.nativeOrder());
}

/** 1. Load your model. */

/** 2. Transform data. */
protected void loadAndProcessBitmap(Bitmap rgbFrameBitmap) {
    Bitmap croppedBitmap = Bitmap.createBitmap(classifier.getImageSizeX(),
        classifier.getImageSizeY(), Config.ARGB_8888);
    final Canvas canvas = new Canvas(croppedBitmap);
    canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);
    imgData.rewind();
    croppedBitmap.getPixels(intValues, 0,
        bitmap.getWidth(), 0, 0, bitmap.getHeight());
    for (int i = 0, pixel = 0; i < getImageSizeX(); ++i) {
        for (int j = 0; j < getImageSizeY(); ++j) {
            final int val = intValues[pixel++];
            imgData.putFloat(((val >> 16) & 0xFF) - IMAGE_MEAN) / IMAGE_STD;
            imgData.putFloat(((val >> 8) & 0xFF) - IMAGE_MEAN) / IMAGE_STD;
            imgData.putFloat((val & 0xFF) - IMAGE_MEAN) / IMAGE_STD;
        }
    }
}

/** 3. Run inference. */
public List<Classification> classifyImage(Bitmap rgbFrameBitmap) {
    loadAndProcessBitmap(rgbFrameBitmap)
    tflite.run(imgData, labelProbArray);
}

/** 3. Use the resulting output. */
PriorityQueue<Classification> pq = new PriorityQueue<Classification>(
    3, new Comparator<Classification>() {
        public int compare(Classification lhs, Classification rhs) {
            return Float.compare(rhs.getConfidence(), lhs.getConfidence());
        }
    });
    for (int i = 0; i < labels.size(); ++i) {
        pq.add(new Classification(
            "" + i, labels.size() > i ? labels.get(i) : "unknown",
            getNormalizedProbability(i)));
    }
    final ArrayList<Classification> results =
        new ArrayList<Classification>();
    int resultSize = Math.min(pq.size(), MAX_RESULTS);
    for (int i = 0; i < resultSize; ++i) {
        results.add(pq.poll());
    }
    return results;
/** With TensorFlow Lite Support Library */

// 1. Load your model.
MyImageClassifier classifier = new MyImageClassifier(activity);
MyImageClassifier.Inputs inputs = classifier.createInputs();

// 2. Transform your data.
inputs.loadImage(rgbFrameBitmap);

MyImageClassifier.Outputs outputs = classifier.run(inputs);

// 4. Use the resulting output.
Map<String, float> labeledProbabilities = outputs.getOutput():
Running Your Model

- Converter
- Interpreter
- Op Kernels
- Delegates
Language Bindings

- New language bindings (Swift, Obj-C, C# and C) for iOS, Android and Unity
- Community language bindings (Rust, Go, Flutter/Dart)
Running TensorFlow Lite on Microcontrollers
What are they?

Small computer on a single circuit

- No operating system
- Tens of KB of RAM & Flash
- Only CPU, memory & I/O peripherals
- Exist all around us
MCU
Is there any sound?

Is that human speech?
TensorFlow Lite for microcontrollers

TensorFlow provides you with a single framework to deploy on Microcontrollers as well as phones
What can you do on an MCU?

- Simple speech recognition
- Person detection using a camera
- Gesture recognition using an accelerometer
- Predictive maintenance
Speech Detection on an MCU

- Recognizes “Yes” and “No”
- Retrainable for other words
- 20KB model
- 7 million ops per second
Person Detection on an MCU

- Recognizes if a person is visible in camera feed
- Retrainable for other objects
- 250KB MobileNet model
- 60 million ops per inference
Gesture Detection on an MCU

- Spots wand gestures
- Retrainable for other gestures
- 20KB model
Improving your model performance
Incredible Performance

Enable your models to run as fast as possible on all hardware
Incredible Performance

CPU
37 ms
Floating point

CPU 2.8x
13 ms
Quantized Fixed-point

GPU 6.2x
6 ms
OpenCL Float16

EdgeTPU 18.5x
2 ms
Quantized Fixed-point

Mobilenet V1

Pixel 4 - Single Threaded CPU, October 2019
Common techniques to improve model performance

- Use quantization
- Use pruning
- Leverage hardware accelerator
- Use mobile optimized model architecture
- Per-op profiling
Utilizing quantization for CPU, DSP & NPU optimizations

Reduce precision of static parameters (e.g. weights) and dynamic values (e.g. activations)
Pruning

Remove connections during training in order to increase sparsity.
Running Your Model

Converter  Interpreter  Op Kernels  Delegates

- Highly optimized for ARM Neon instruction set
- Accelerators like GPU, DSP and Edge TPU
- Integrate with Android Neural Network API
Utilizing Accelerators via Delegates

CPU Operation Kernels

Interpreter Core

Accelerator Delegate

Op

Op

Input

Op

Op

Activation

Op
GPU Delegation enables faster float execution

- 2–7x faster than the floating point CPU implementation
- Uses OpenGL & OpenCL on Android and Metal on iOS
- Accepts float models (float16 or float32)
DSP Delegation through Qualcomm Hexagon DSP

- Use Hexagon delegate on Android O & below
- Use NN API on Android P & beyond
- Accepts integer models (uint8)
- Launching soon!
Delegation through Android Neural Networks API

- Enables graph acceleration on DSP, GPU and NPU
- Supports 30+ ops in Android P, 90+ ops in Android Q
- Accepts float (float16, float32) and integer models (uint8)
/** Initializes an {@code ImageClassifier}. */

ImageClassifier(Activity activity) throws IOException {
    tfliteModel = loadModelFile(activity);

    delegate = new GpuDelegate();
    tfliteOptions.addDelegate(delegate);

    tflite = new Interpreter(tfliteModel, tfliteOptions);
    ...
}
/** Initializes an {@code ImageClassifier}. */
ImageClassifier(Activity activity) throws IOException {
    tfliteModel = loadModelFile(activity);

delegate = new NnApiDelegate();
tfliteOptions.addDelegate(delegate);

tflite = new Interpreter(tfliteModel, tfliteOptions);
...
}
## Model Comparison

<table>
<thead>
<tr>
<th></th>
<th>Inception v3</th>
<th>Mobilenet v1</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top-1 accuracy</strong></td>
<td>77.9%</td>
<td>68.3%</td>
<td>-11%</td>
</tr>
<tr>
<td><strong>Top-5 accuracy</strong></td>
<td>93.8%</td>
<td>88.1%</td>
<td>-6%</td>
</tr>
<tr>
<td><strong>Inference latency</strong></td>
<td>1433 ms</td>
<td>95.7 ms</td>
<td>15x faster</td>
</tr>
<tr>
<td><strong>Model size</strong></td>
<td>95.3 MB</td>
<td>10.3 MB</td>
<td>9.3x smaller</td>
</tr>
</tbody>
</table>
bazel build -c opt \
  --config=android_arm64 --cxxopt='--std=c++11' \ 
  --copt=-DTFLITE_PROFILING_ENABLED \ 
  //tensorflow/lite/tools/benchmark:benchmark_model

adb push .../benchmark_model /data/local/tmp
adb shell taskset f0 /data/local/tmp/benchmark_model
Per-op Profiling

Number of nodes executed: 31

<table>
<thead>
<tr>
<th>[node type]</th>
<th>[count]</th>
<th>[avg ms]</th>
<th>[avg %]</th>
<th>[cdf %]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV_2D</td>
<td>15</td>
<td>1.406</td>
<td>89.270%</td>
<td>89.270%</td>
</tr>
<tr>
<td>DEPTHWISE_CONV_2D</td>
<td>13</td>
<td>0.169</td>
<td>10.730%</td>
<td>100.000%</td>
</tr>
<tr>
<td>SOFTMAX</td>
<td>1</td>
<td>0.000</td>
<td>0.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>RESHAPE</td>
<td>1</td>
<td>0.000</td>
<td>0.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>AVERAGE_POOL_2D</td>
<td>1</td>
<td>0.000</td>
<td>0.000%</td>
<td>100.000%</td>
</tr>
</tbody>
</table>
Improving your **operator** coverage
Expand operators, reduce size

- Utilize TensorFlow ops if op is not natively supported
- Only include required ops to reduce the runtime’s size
Using TensorFlow operators

- Enables hundreds more ops from TensorFlow on CPU
- Caveat: Binary size increase (~6MB compressed)
import tensorflow as tf

c = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

c.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
                               tf.lite.OpsSet.SELECT_TF_OPS]

tflite_model = c.convert()
open("converted_model.tflite", "wb").write(tflite_model)
Reduce overall runtime size

- Selectively include only the ops required by the model
- Pares down the size of the binary
/* my_inference.cc */

// Forward declaration for RegisterSelectedOps.
void RegisterSelectedOps(::tflite::MutableOpResolver* resolver);

::tflite::MutableOpResolver resolver;
RegisterSelectedOps(&resolver);

std::unique_ptr<::tflite::Interpreter> interpreter;
::tflite::InterpreterBuilder(*model, resolver)(&interpreter);

...
gen_selected_ops(
    name = "my_op_resolver"
    model = ":my_tflite_model"
)

cc_library(
    name = "my_inference",
    srcs = ["my_inference.cc", ":my_op_resoler"]
)
How to get started
Brand new course launched on Udacity for TensorFlow Lite
Intro to embedded deep learning with TensorFlow Lite
Monthly meetups on embedded ML

- Santa Clara
- Austin
- More coming soon!

tinyurl.com/tinyml-santaclara
tinyurl.com/tinyml-austin
Visit tensorflow.org/lite

TensorFlow Lite

Questions?
tflite@tensorflow.org