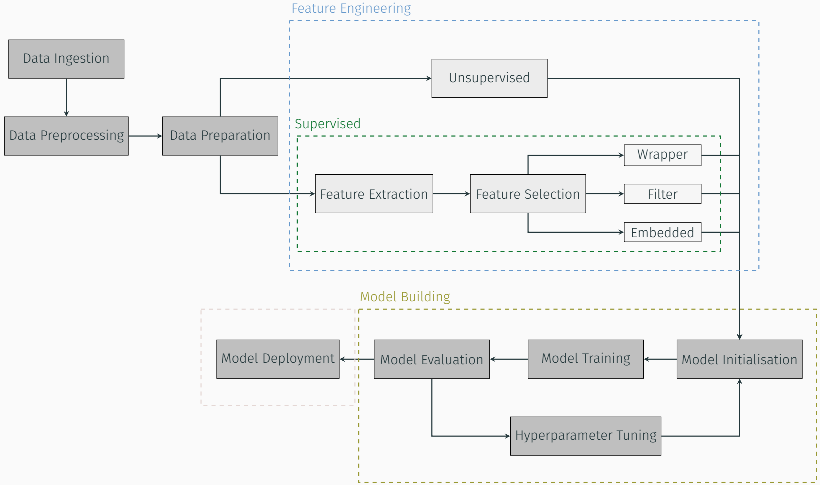
Introduction to TensorFlow Lite

Madeira International Workshop in Machine Learning

Diogo Freitas

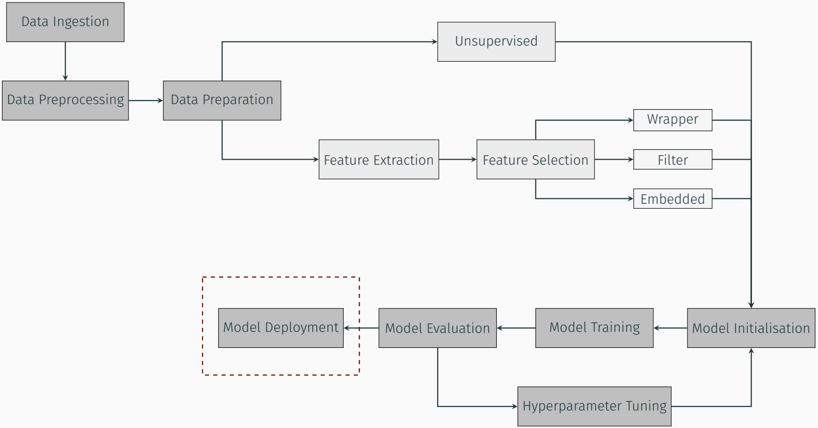
July 30, 2021

Where are we now?



Now we want to deploy our model! ➔ TensorFlow Lite

Where are we now?



Now we want to deploy our model! ➜ TensorFlow Lite

Wait...


DEPLOY ON A FRIDAY THEY SAID
38
IT'LL BE FINE THEY SAID
makeameme.org

Why TensorFlow Lite?

- TensorFlow Lite enables on–device machine learning (e.g., Android and iOS devices, and microcontrollers).
- TensorFlow models can be used without an internet connection.
- Enables to create lightweight models with low latency (i.e., with high performance).
- Includes techniques for hardware acceleration and model optimization.
- Low power consumption.

Why TensorFlow Lite?

- TensorFlow Lite enables on–device machine learning (e.g., Android and iOS devices, and microcontrollers).
- TensorFlow models can be used without an internet connection.
- Enables to create lightweight models with low latency (i.e., with high performance).
- Includes techniques for hardware acceleration and model optimization.
- Low power consumption.

Why TensorFlow Lite?

- TensorFlow Lite enables on–device machine learning (e.g., Android and iOS devices, and microcontrollers).
- TensorFlow models can be used without an internet connection.
- Enables to create lightweight models with low latency (i.e., with high performance).
- Includes techniques for hardware acceleration and model optimization.
- Low power consumption.

Why TensorFlow Lite?

- TensorFlow Lite enables on–device machine learning (e.g., Android and iOS devices, and microcontrollers).
- TensorFlow models can be used without an internet connection.
- Enables to create lightweight models with low latency (i.e., with high performance).
- Includes techniques for hardware acceleration and model optimization.
- Low power consumption.

Why TensorFlow Lite?

- TensorFlow Lite enables on–device machine learning (e.g., Android and iOS devices, and microcontrollers).
- TensorFlow models can be used without an internet connection.
- Enables to create lightweight models with low latency (i.e., with high performance).
- Includes techniques for hardware acceleration and model optimization.
- Low power consumption.

1

Build a model

2

Export and convert

3

Verify

4

Deploy

1

Build a model

2

Export and convert

3

Verify

4

Deploy

Build a model

We already know how to build a model, but let us review it.

1. Create some dummy data (for demonstration purposes only):

```
X = np.array([1, 2, 3, 4, 5, 6])
y = np.array([3, 5, 7, 9, 11, 13])
```

2. Create the model with a single hidden unit:

```
model = tf.keras.models.Sequential(
        [tf.keras.layers.InputLayer(input_shape=(1,)),
         tf.keras.layers.Dense(units=1)]
        )
```

Build a model

We already know how to build a model, but let us review it.

1. Create some dummy data (for demonstration purposes only):

```
X = np.array([1, 2, 3, 4, 5, 6])
y = np.array([3, 5, 7, 9, 11, 13])
```

2. Create the model with a single hidden unit:

```
model = tf.keras.models.Sequential(
        [tf.keras.layers.InputLayer(input_shape=(1,)),
         tf.keras.layers.Dense(units=1)]
        )
```

Build a model (cont.)

3. Compile the model:

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

4. Train the model:

```
model.fit(X, y, epochs=10)
```

Build a model (cont.)

3. Compile the model:

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

4. Train the model:

```
model.fit(X, y, epochs=10)
```

1. Build a model
2. Export and convert
3. Verify
4. Deploy

Export and convert

You can generate a TensorFlow Lite model by:

- using an existing TensorFlow Lite model.
- creating a TensorFlow Lite model.
- converting a TensorFlow model into a TensorFlow Lite model.

File model extension:

Files generated by TensorFlow Lite are identified by the `.tflite` file extension.

Export and convert

You can generate a TensorFlow Lite model by:

- using an existing TensorFlow Lite model.
- creating a TensorFlow Lite model.
- converting a TensorFlow model into a TensorFlow Lite model.

File model extension:

Files generated by TensorFlow Lite are identified by the `.tflite` file extension.

Export and convert

You can generate a TensorFlow Lite model by:

- using an existing TensorFlow Lite model.
- creating a TensorFlow Lite model.
- converting a TensorFlow model into a TensorFlow Lite model.

File model extension:

Files generated by TensorFlow Lite are identified by the `.tflite` file extension.

Export and convert

You can generate a TensorFlow Lite model by:

- using an existing TensorFlow Lite model.
- creating a TensorFlow Lite model.
- converting a TensorFlow model into a TensorFlow Lite model.

File model extension:

Files generated by TensorFlow Lite are identified by the `.tflite` file extension.

Components of TensorFlow Lite

Converter

- Takes a TensorFlow model and generates a TensorFlow Lite model.
- Enables to export the model with various optimizations options and various platforms.

Interpreter

- Enables to run inference (i.e., given the inputs, compute the model's output) on a client device.
- Enables to run the model in various platforms (e.g., Linux and iOS).
- Provides hardware-accelerated APIs.

Components of TensorFlow Lite

Converter

- Takes a TensorFlow model and generates a TensorFlow Lite model.
- Enables to export the model with various optimizations options and various platforms.

Interpreter

- Enables to run inference (i.e., given the inputs, compute the model's output) on a client device.
- Enables to run the model in various platforms (e.g., Linux and iOS).
- Provides hardware-accelerated APIs.

1. Export the *SavedModel*[1]:

```
export_dir = '/tmp/saved_model/1'
tf.saved_model.save(model, export_dir=export_dir)
```

2. Convert the model:

```
converter = tf.lite.TFLiteConverter.from_saved_model(
    export_dir)
tflite_model = converter.convert()
```

3. Save the model:

```
tflite_model_file = pathlib.Path('/tmp/model.tflite')
tflite_model_file.write_bytes(tflite_model)
```

[1] This is the standard for serializing a TensorFlow model.

Export and convert (cont.)

1. Export the *SavedModel*[1]:

```
export_dir = '/tmp/saved_model/1'
tf.saved_model.save(model, export_dir=export_dir)
```

2. Convert the model:

```
converter = tf.lite.TFLiteConverter.from_saved_model(
    export_dir)
tflite_model = converter.convert()
```

3. Save the model:

```
tflite_model_file = pathlib.Path('/tmp/model.tflite')
tflite_model_file.write_bytes(tflite_model)
```

[1] This is the standard for serializing a TensorFlow model.

Export and convert (cont.)

1. Export the *SavedModel*[1]:

```
export_dir = '/tmp/saved_model/1'
tf.saved_model.save(model, export_dir=export_dir)
```

2. Convert the model:

```
converter = tf.lite.TFLiteConverter.from_saved_model(
    export_dir)
tflite_model = converter.convert()
```

3. Save the model:

```
tflite_model_file = pathlib.Path('/tmp/model.tflite')
tflite_model_file.write_bytes(tflite_model)
```

[1] This is the standard for serializing a TensorFlow model.

1. Build a model
2. Export and convert
3. Verify
4. Deploy

Verify

TensorFlow Lite has a built-in interpreter in Python. You can test your model before deployment.

1. Load the TensorFlow Lite model and allocate tensors:

```
interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
```

2. Get input and output tensors:

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

3. Test the model on input data:

```
interpreter.set_tensor(input_details[0]['index'],
    input_data)
```

Verify

TensorFlow Lite has a built-in interpreter in Python. You can test your model before deployment.

1. Load the TensorFlow Lite model and allocate tensors:

```
interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
```

2. Get input and output tensors:

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

3. Test the model on input data:

```
interpreter.set_tensor(input_details[0]['index'],
    input_data)
```

Verify

TensorFlow Lite has a built-in interpreter in Python. You can test your model before deployment.

1. Load the TensorFlow Lite model and allocate tensors:

```
interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
```

2. Get input and output tensors:

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

3. Test the model on input data:

```
interpreter.set_tensor(input_details[0]['index'],
    input_data)
```

4. Run the interpreter

```
interpreter.invoke()
```

5. Get the output data:

```
output_data = interpreter.get_tensor(output_details[0]['
    index'])
```

Verify (cont.)

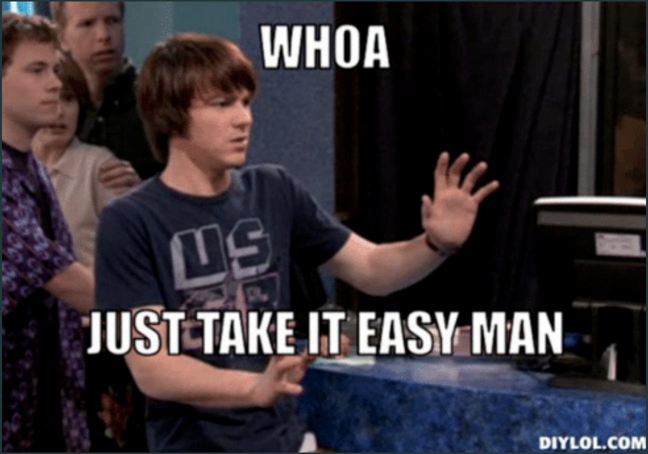
4. Run the interpreter

```
interpreter.invoke()
```

5. Get the output data:

```
output_data = interpreter.get_tensor(output_details[0]['
    index'])
```

DIY. Start with the notebook provided on GitHub.



☑ You can now use the model on your device!

1. Build a model
2. Export and convert
3. Verify
4. **Deploy**

Optimization

Optimization might be necessary due to limited resources.

With optimization, one wants to reduce latency and the power consumption of the models.

Solution ➔ Quantization!

Quantization

Is an optimization technique that reduces the precision of the numbers in the weights and biases of the model, but at the same time, reduces model size and improves CPU and hardware accelerator latency.

You will notice, however, a little degradation in the model's accuracy!

Optimization

Optimization might be necessary due to limited resources.

With optimization, one wants to reduce latency and the power consumption of the models.

Solution ➔ Quantization!

Quantization

Is an optimization technique that reduces the precision of the numbers in the weights and biases of the model, but at the same time, reduces model size and improves CPU and hardware accelerator latency.

You will notice, however, a little degradation in the model's accuracy!

Optimization

Optimization might be necessary due to limited resources.

With optimization, one wants to reduce latency and the power consumption of the models.

Solution ➔ Quantization!

Quantization

Is an optimization technique that reduces the precision of the numbers in the weights and biases of the model, but at the same time, reduces model size and improves CPU and hardware accelerator latency.

You will notice, however, a little degradation in the model's accuracy!

Optimization

Optimization might be necessary due to limited resources.

With optimization, one wants to reduce latency and the power consumption of the models.

Solution ➜ Quantization!

Quantization

Is an optimization technique that reduces the precision of the numbers in the weights and biases of the model, but at the same time, reduces model size and improves CPU and hardware accelerator latency.

You will notice, however, a little degradation in the model's accuracy!

Types of quantization

Dynamic

Quantizes only the weights from floating-point to integer, which has 8-bits of precision. However, inference operations are still done in floating-point precision.

Full integer quantization

This technique makes all the model's math integer quantized. However, it uses float operators when they don't have an integer implementation.

Integer only

Only used to ensure compatibility with integer only devices (such as 8-bit microcontrollers) by enforcing full-integer quantization for all operations including the input and output.

Types of quantization

Dynamic

Quantizes only the weights from floating-point to integer, which has 8-bits of precision. However, inference operations are still done in floating-point precision.

Full integer quantization

This technique makes all the model's math integer quantized. However, it uses float operators when they don't have an integer implementation.

Integer only

Only used to ensure compatibility with integer only devices (such as 8-bit microcontrollers) by enforcing full-integer quantization for all operations including the input and output.

Types of quantization

Dynamic

Quantizes only the weights from floating-point to integer, which has 8-bits of precision. However, inference operations are still done in floating-point precision.

Full integer quantization

This technique makes all the model's math integer quantized. However, it uses float operators when they don't have an integer implementation.

Integer only

Only used to ensure compatibility with integer only devices (such as 8-bit microcontrollers) by enforcing full-integer quantization for all operations including the input and output.

Quantization in code

You can specify the type of quantization technique that you want to use for your model in the converter options.

- Dynamic

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

- Full integer quantization

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset
```

- Integer only

```
converter.target_spec.supported_ops = [tf.lite.OpsSet.
    TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
```

Quantization in code

You can specify the type of quantization technique that you want to use for your model in the converter options.

- Dynamic

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

- Full integer quantization

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset
```

- Integer only

```
converter.target_spec.supported_ops = [tf.lite.OpsSet.
    TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
```

Quantization in code

You can specify the type of quantization technique that you want to use for your model in the converter options.

- Dynamic

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

- Full integer quantization

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset
```

- Integer only

```
converter.target_spec.supported_ops = [tf.lite.OpsSet.
    TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
```

Quantization in code (cont.)

It's a small dataset to calibrate or estimate the range, i.e (min, max), of all floating-point arrays in the model for quantization.

This dataset can be a small subset (around 100–500 samples) of the training or validation data.

This dataset is generated using a generator function (remember the first day of the workshop?), such as:

```
def representative_dataset():
    for _ in range(100):
        data = x_train[:100]
        yield [tf.dtypes.cast(data, tf.float32)]
```

Quantization in code (cont.)

It's a small dataset to calibrate or estimate the range, i.e (min, max), of all floating-point arrays in the model for quantization.

This dataset can be a small subset (around 100–500 samples) of the training or validation data.

This dataset is generated using a generator function (remember the first day of the workshop?), such as:

```
def representative_dataset():
    for _ in range(100):
        data = x_train[:100]
        yield [tf.dtypes.cast(data, tf.float32)]
```

Quantization in code (cont.)

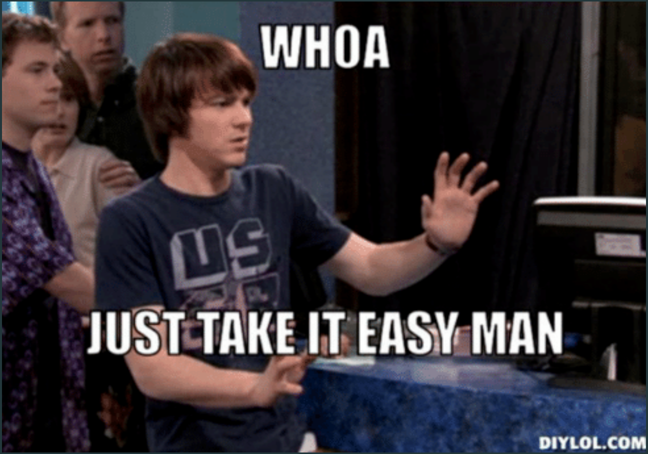
It's a small dataset to calibrate or estimate the range, i.e (min, max), of all floating-point arrays in the model for quantization.

This dataset can be a small subset (around 100–500 samples) of the training or validation data.

This dataset is generated using a generator function (remember the first day of the workshop?), such as:

```
def representative_dataset():
    for _ in range(100):
        data = x_train[:100]
        yield [tf.dtypes.cast(data, tf.float32)]
```

DIY. Start with the notebook provided on GitHub.



REST API

TensorFlow Lite has still some limitations in terms of operator compatibility.

Some models are still relatively too big to store on devices.

One solution ➔ Execute the model in a remote server using a REST API!

REST API

TensorFlow Lite has still some limitations in terms of operator compatibility.

Some models are still relatively too big to store on devices.

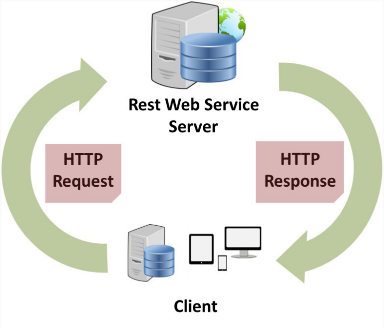
One solution ➔ Execute the model in a remote server using a REST API!

REST API

TensorFlow Lite has still some limitations in terms of operator compatibility.

Some models are still relatively too big to store on devices.

One solution ➔ Execute the model in a remote server using a REST API!



REST API (cont.)

- HTTP request by the client:

```
GET workshop/?input= and hell!\nAll hurt behind; backs red, and faces pale\nWith f
```

- HTTP response by the remote server:

```
{
    "input": " and hell!\\nAll hurt behind; backs red, and faces pale\\nWith f",
    "output": " and hell!\nAll hurt behind; backs red, and faces pale\nWith fake wores theme your fore is man, no 'sbers; bays?\nGut that tho becues! Grown\nwet as a teace paolly, hich.\n\nQUEEs I' meer theing that.\nHe suatt mo carknsed suchenics, for Ratherrshads;\nIn worvake furb\n"
}
```

REST API (cont.)

- HTTP request by the client:

```
GET workshop/?input= and hell!\nAll hurt behind; backs red,
    and faces pale\nWith f
```

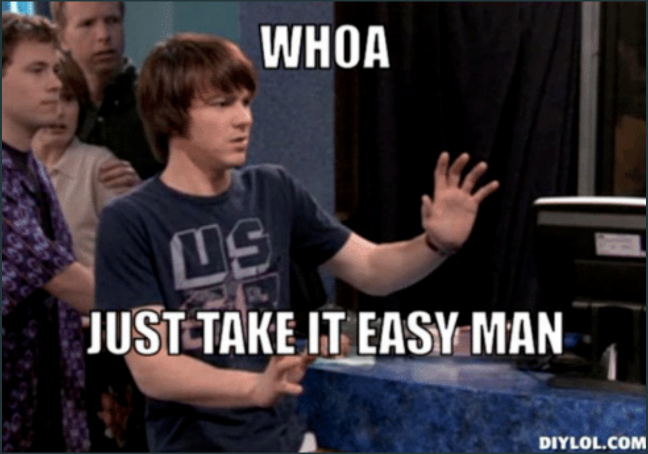
- HTTP response by the remote server:

```
{
    "input": " and hell!\\nAll hurt behind; backs red, and
    faces pale\\nWith f",
    "output": " and hell!\nAll hurt behind; backs red, and
    faces pale\nWith fake wores theme your fore is man, no
    'sbers; bays?\nGut that tho becues! Grown\nwet as a
    teace paolly, hich.\n\nQUEEs I' meer theing that.\nHe
    suatt mo carknsed suchenics, for Ratherrshads;\nIn
    worvake furb\n"
}
```

A status code is issued by the remote server in response to a client's request made to the server.

Code	Status
200	OK
400	Bad request
404	Not found
500	Internal server error
⋮	⋮

DIY. Start with the notebook provided on GitHub.



Transfer Learning with TensorFlow Hub

TensorFlow Hub (https://tfhub.dev) is a repository of trained machine learning models ready to use.

These models can be then fine-tunned for a specific problem and deployed anywhere using, e.g., using TensorFlow Lite.

With transfer learning, a model developed for a specific task can reused as the starting point (e.g., for feature extraction) for other model on a different task.

Transfer Learning with TensorFlow Hub

TensorFlow Hub (https://tfhub.dev) is a repository of trained machine learning models ready to use.

These models can be then fine-tunned for a specific problem and deployed anywhere using, e.g., using TensorFlow Lite.

With transfer learning, a model developed for a specific task can reused as the starting point (e.g., for feature extraction) for other model on a different task.

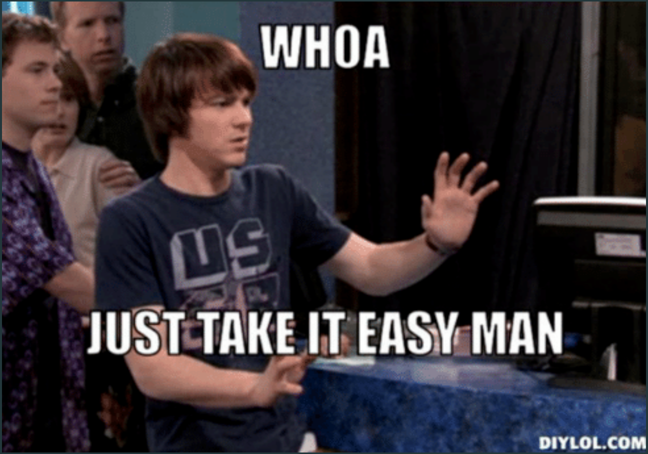
Transfer Learning with TensorFlow Hub

TensorFlow Hub (https://tfhub.dev) is a repository of trained machine learning models ready to use.

These models can be then fine-tunned for a specific problem and deployed anywhere using, e.g., using TensorFlow Lite.

With transfer learning, a model developed for a specific task can reused as the starting point (e.g., for feature extraction) for other model on a different task.

DIY. Start with the notebook provided on .



Questions?