Copyright Notice

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Features

Lightweight

Low-latency

Privacy

Improved power consumption

Efficient model format

Pre-trained models

Components in TensorFlow Lite

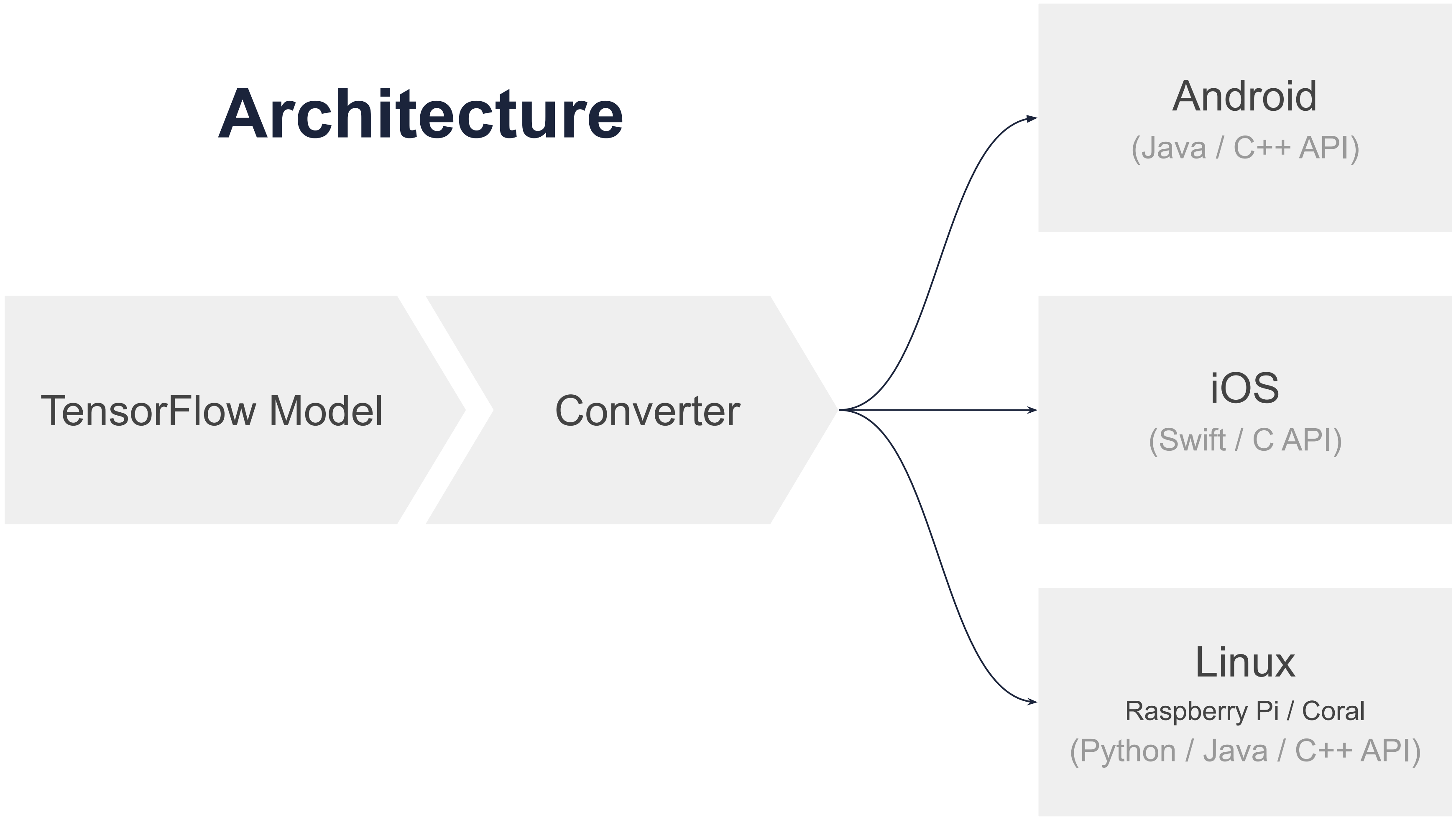
Converter
(to TensorFlow Lite format)

- Transforms TensorFlow models into a form efficient for reading by the interpreter
- Introduces optimizations to improve binary size model performance and/or reduce model size.

Interpreter
Core

- Diverse platform support, (Android, iOS, embedded Linux and microcontrollers)
- Platform APIs for accelerated inference

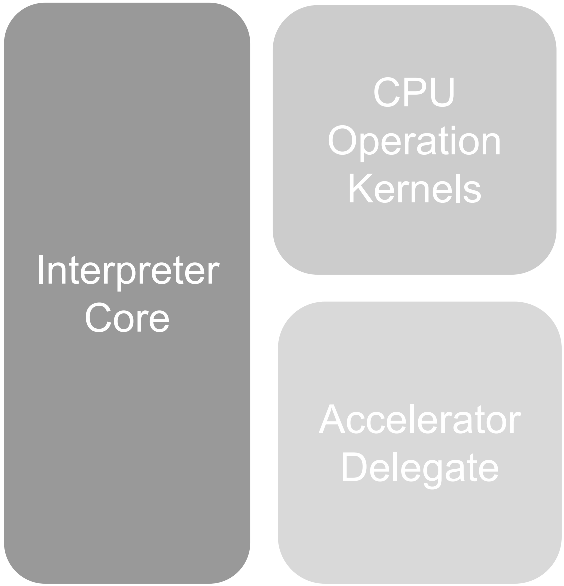
Architecture

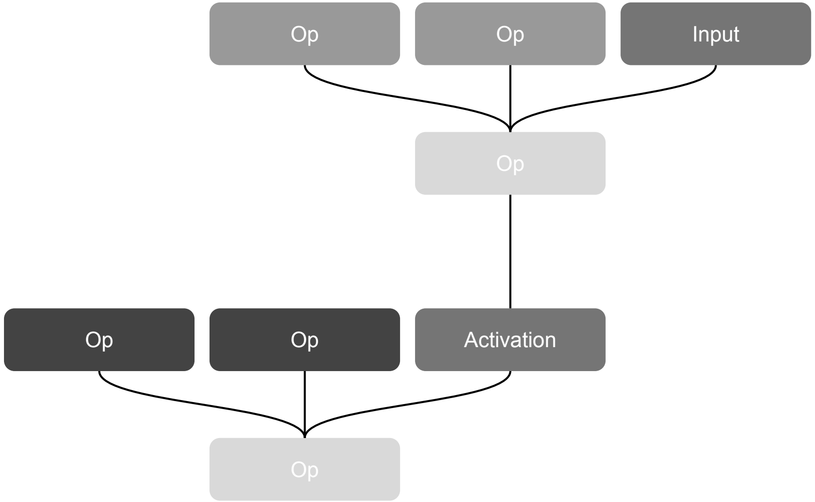


Performance

Acceleration	Available
Software	NN API (also a delegate)
Hardware	Edge TPU
	GPU
	CPU Optimizations (ARM and x86)

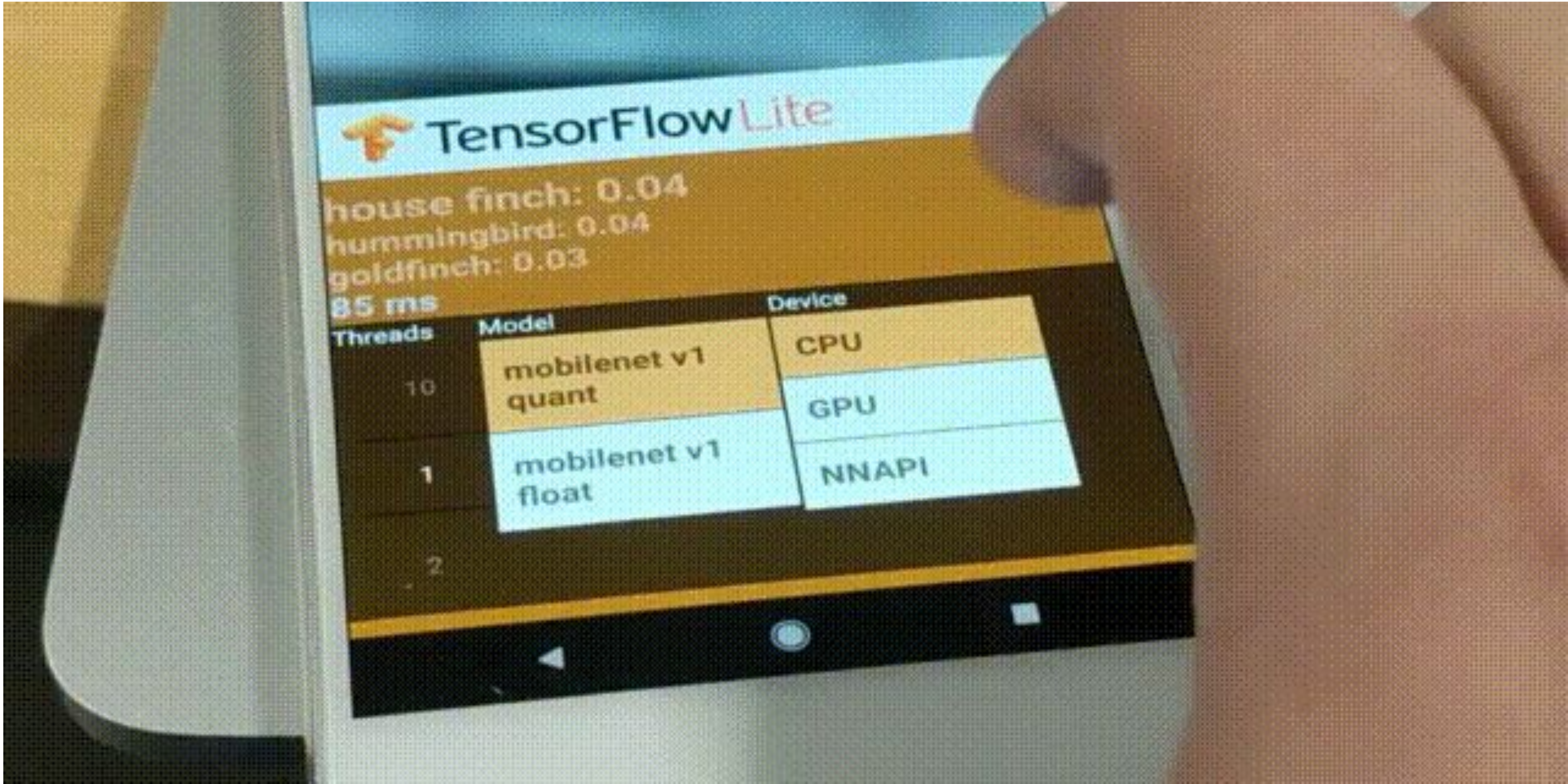
Delegates


Interpreter
Core
CPU
Operation
Kernels
Accelerator
Delegate


Op
Op
Input
Op
Op
Op
Activation
Op

https://www.youtube.com/watch?v=QSbAUxWfxQw

https://www.tensorflow.org/lite/performance/gpu



Techniques

- Quantization
- Weight pruning
- Model topology transforms
  - Tensor Decomposition
  - Distillation

Why Quantize?

- All available CPU platforms are supported
- Reducing latency and inference cost
- Low memory footprint
- Allow execution on hardware restricted-to or optimized-for fixed-point operations
- Optimized models for special purpose HW accelerators (TPUs)

Putting it all together

1 — 2 — 3 — 4

1. Jump start

Use Pretrained or Retrained Models

2. Custom model

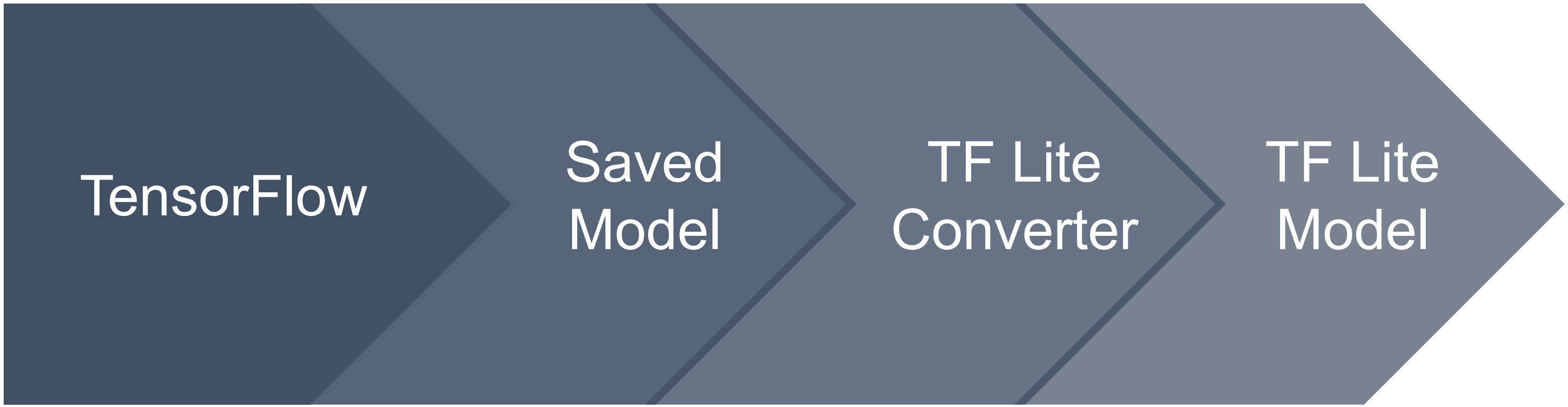
Develop & deploy a custom model

3. Performance

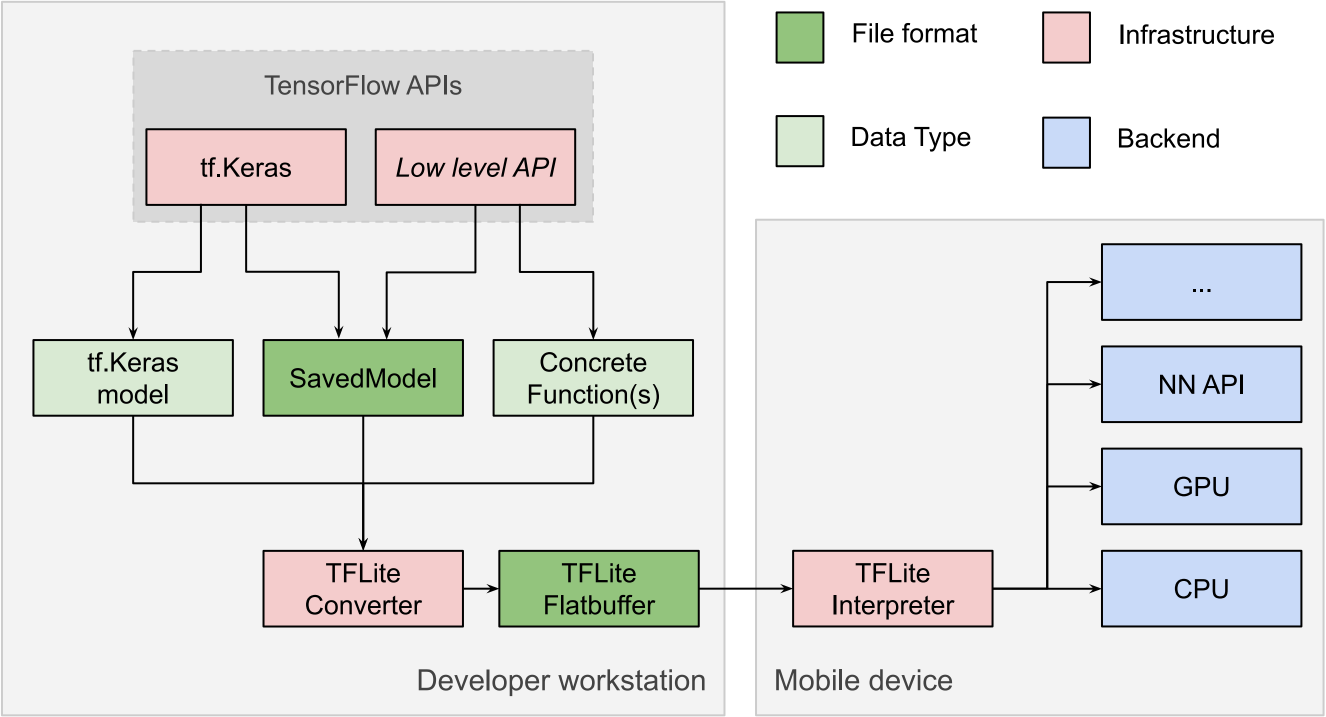
Explore options, validate & accelerate models.

4. Optimize

Model Optimization Toolkit


TensorFlow
Saved Model
TF Lite Converter
TF Lite Model

TensorFlow Lite Converter



Parameters for conversion

Inputs

SavedModel (preferred)	`tf.lite.TFLiteConverter.from_saved_model(...)`	
Keras a model instance HDF5 file	`tf.lite.TFLiteConverter.from_keras_model(...)`	**Output** TFLite model
Concrete function(s)	`tf.lite.TFLiteConverter.from_concrete_functions(...)`	

Python API (preferred)

SavedModel

- The standard for serializing a TensorFlow model
- A MetaGraph to hold metadata
- Holds snapshot of the trained model (with model weights and computation)
- No model building code required
- Supports model versioning

Inspecting with SavedModel's CLI

```
!saved_model_cli show --dir /tmp/mobilenet/1 \
                      --tag_set serve \
                      --signature_def serving_default
```

Inspecting with SavedModel's CLI

```
The given SavedModel SignatureDef contains the following input(s):
  inputs['input_1'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 224, 224, 3)
      name: serving_default_input_1:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['act_softmax'] tensor_info:
      dtype: DT_FLOAT
      shape: (-1, 1000)
      name: StatefulPartitionedCall:0
Method name is: tensorflow/serving/predict
```

Exporting a SavedModel from Keras

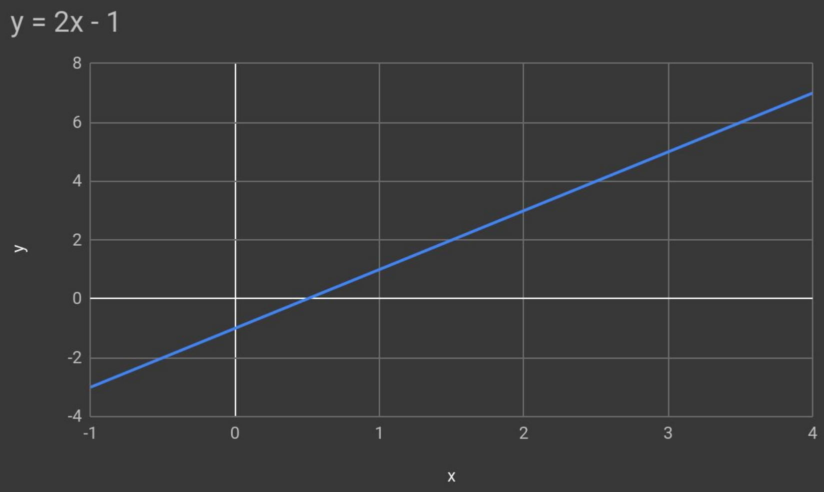
```
pretrained_model = tf.keras.applications.MobileNet()
tf.saved_model.save(pretrained_model, '/tmp/saved_model/1/')
```

Example 1


```
import tensorflow as tf

# Store data for x and y
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]


# Create a simple Keras model.
model = tf.keras.models.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(x, y, epochs=500)
```



Example 1


```
import pathlib

# Export the SavedModel
export_dir = '/tmp/saved_model'
tf.saved_model.save(model, export_dir)

# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 1


```python
import pathlib

# Export the SavedModel
export_dir = '/tmp/saved_model'
tf.saved_model.save(model, export_dir)

# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 1


```
import pathlib

# Export the SavedModel
export_dir = '/tmp/saved_model'
tf.saved_model.save(model, export_dir)

# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 1


```
import pathlib

# Export the SavedModel
export_dir = '/tmp/saved_model'
tf.saved_model.save(model, export_dir)

# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 2


```python
import tensorflow as tf
import pathlib

# Load the MobileNet tf.keras model.
model = tf.keras.applications.MobileNetV2(weights="imagenet", input_shape=(224, 224, 3))
# Saving the model for later use by tflite_convert
model.save('model.h5')

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 2


```
import tensorflow as tf
import pathlib

# Load the MobileNet tf.keras model.
model = tf.keras.applications.MobileNetV2(weights="imagenet", input_shape=(224, 224, 3))
# Saving the model for later use by tflite_convert
model.save('model.h5')

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 2


```
import tensorflow as tf
import pathlib

# Load the MobileNet tf.keras model.
model = tf.keras.applications.MobileNetV2(weights="imagenet", input_shape=(224, 224, 3))
# Saving the model for later use by tflite_convert
model.save('model.h5')

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Example 2


```
import tensorflow as tf
import pathlib

# Load the MobileNet tf.keras model.
model = tf.keras.applications.MobileNetV2(weights="imagenet", input_shape=(224, 224, 3))
# Saving the model for later use by tflite_convert
model.save('model.h5')

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
tflite_model_file = pathlib.Path('/tmp/foo.tflite')
tflite_model_file.write_bytes(tflite_model)
```

Command-line usage

```
#!/usr/bin/env bash

# Saving with the command-line from a SavedModel
tflite_convert --output_file=model.tflite --saved_model_dir=/tmp/saved_model

# Saving with the command-line from a Keras model
tflite_convert --output_file=model.tflite --keras_model_file=model.h5
```

Command-line usage

```bash
#!/usr/bin/env bash

# Saving with the command-line from a SavedModel
tflite_convert --output_file=model.tflite --saved_model_dir=/tmp/saved_model

# Saving with the command-line from a Keras model
tflite_convert --output_file=model.tflite --keras_model_file=model.h5
```

Command-line usage

```bash
#!/usr/bin/env bash

# Saving with the command-line from a SavedModel
tflite_convert --output_file=model.tflite --saved_model_dir=/tmp/saved_model

# Saving with the command-line from a Keras model
tflite_convert --output_file=model.tflite --keras_model_file=model.h5
```

Post-training quantization

- Reduced precision representation with 3x lower latency
- Little degradation in model accuracy
- Optimization modes
  - ↕ Default (both size and latency)
  - ↓ Size
  - ↓ Latency
- Efficiently represents an arbitrary magnitude of ranges
- Quantization target specification (FP32/INT8)

Image credits

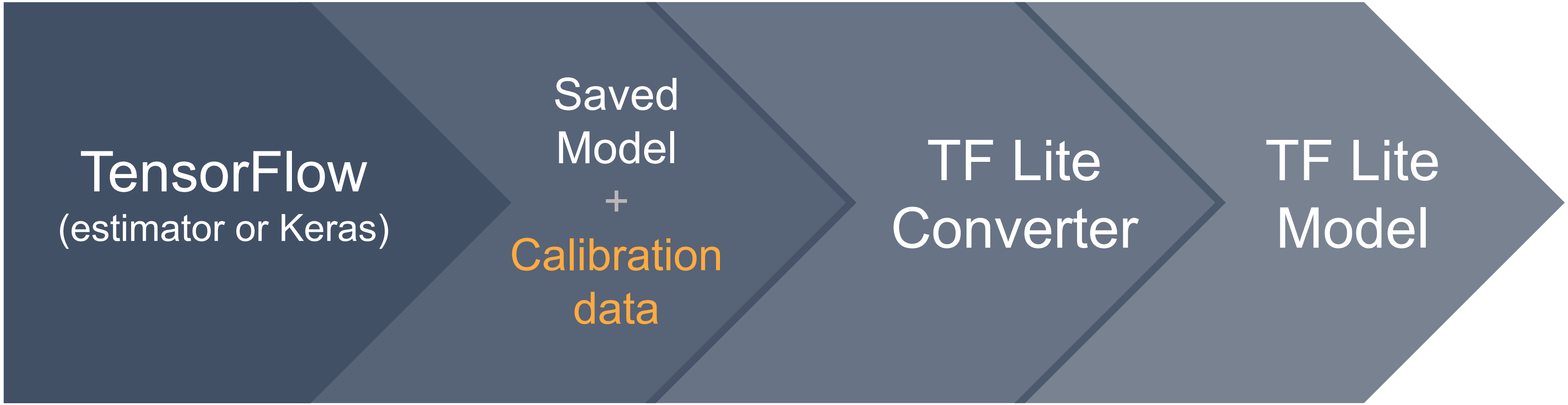
```
import tensorflow as tf

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

converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]

tflite_quant_model = converter.convert()
```

Post-training integer quantization



```
# Define the generator
def generator():
  data = tfds.load(...)
  for _ in range(num_calibration_steps):
    image, = data.take(1)
    yield [image]

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

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to the converter
converter.representative_dataset = tf.lite.RepresentativeDataset(generator)
```

```
# Define the generator
def generator():
  data = tfds.load(...)
  for _ in range(num_calibration_steps):
    image, = data.take(1)
    yield [image]

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

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to the converter
converter.representative_dataset = tf.lite.RepresentativeDataset(generator)
```

```
# Define the generator
def generator():
  data = tfds.load(...)
  for _ in range(num_calibration_steps):
    image, = data.take(1)
    yield [image]

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

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to the converter
converter.representative_dataset = tf.lite.RepresentativeDataset(generator)
```

```
# Define the generator
def generator():
  data = tfds.load(...)
  for _ in range(num_calibration_steps):
    image, = data.take(1)
    yield [image]

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

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to the converter
converter.representative_dataset = tf.lite.RepresentativeDataset(generator)
```

```
...
# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_LATENCY]

# Pass the representative dataset to the converter
converter.representative_dataset = tf.lite.RepresentativeDataset(generator)

# Restricting supported target op specification to INT8
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
```

Learn more about supported ops:

https://www.tensorflow.org/lite/guide/ops_compatibility

TF-Select to overcome unsupported ops

TF-Select

https://www.tensorflow.org/lite/guide/ops_select

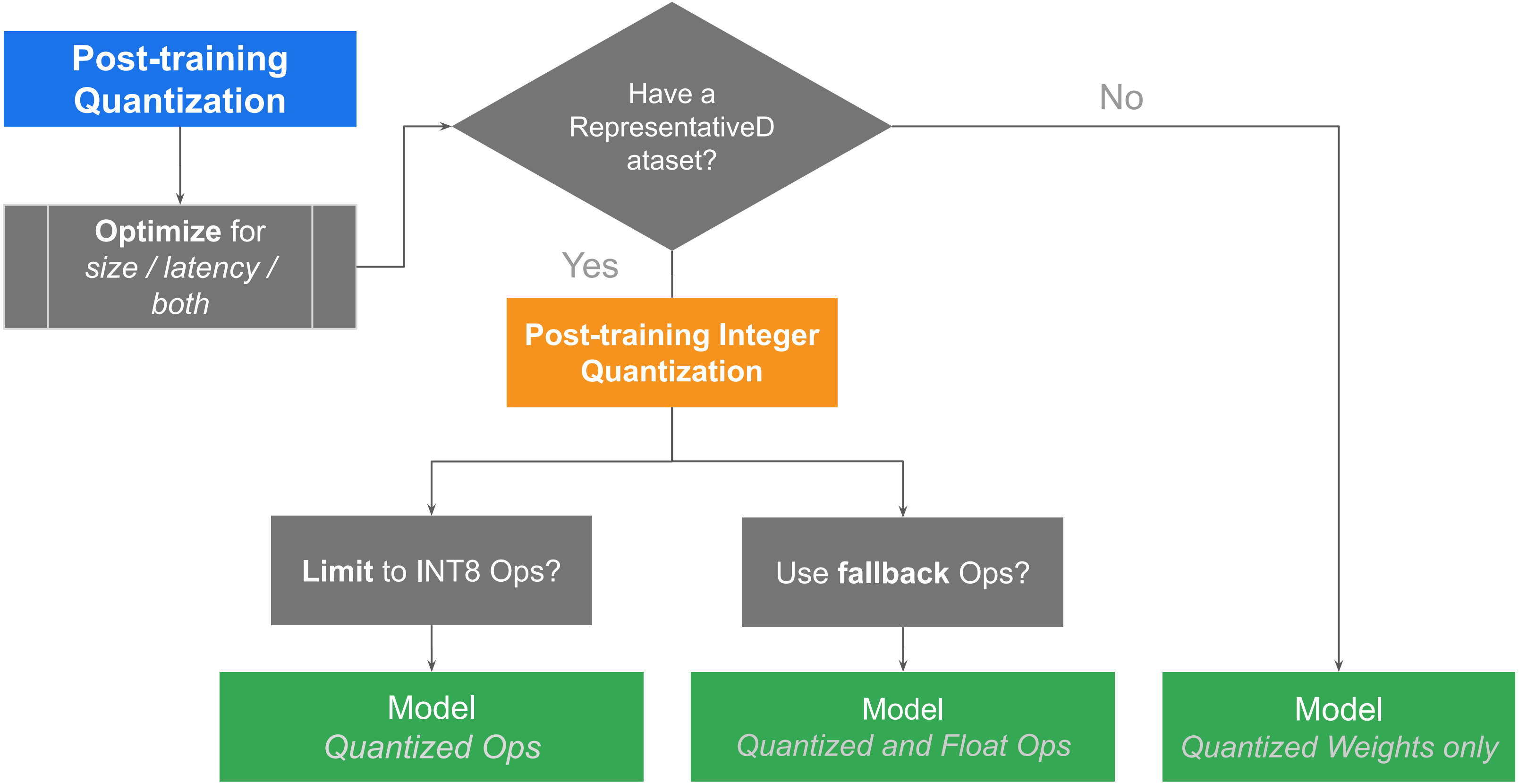
```
import tensorflow as tf

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

converter.target_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
                        tf.lite.OpsSet.SELECT_TF_OPS]

tflite_model = converter.convert()
```

Optimizing your models in a nutshell



TensorFlow Lite Interpreter in Python

```python
# Load TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Point the data to be used for testing and run the interpreter
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

TensorFlow Lite Interpreter in Python

```
# Load TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Point the data to be used for testing and run the interpreter
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

TensorFlow Lite Interpreter in Python

```
# Load TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Point the data to be used for testing and run the interpreter
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

TensorFlow Lite Interpreter in Python

```
# Load TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Point the data to be used for testing and run the interpreter
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

TensorFlow Lite Interpreter in Python

```
# Load TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Point the data to be used for testing and run the interpreter
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
tflite_results = interpreter.get_tensor(output_details[0]['index'])
```

Running models

Pretrained models

Image classification

Object detection

Smart reply

Pose estimation

Segmentation

TensorFlow Hub

Classification modules

Feature vector modules

Embedding modules

Not what you're looking for?

Build a custom model!

Getting a basic model running

Get started

1

Build a model

Create a simple model for (y = 2x - 1) from simulated data and train it

2

Export & Convert

Generate the SavedModel and convert it to TFLite

3

Verify

Perform evaluation on random data to verify the results

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Getting a basic model running

Get started

1

Build a model

Create a simple model for $(y = 2x - 1)$ from simulated data and train it

2

Export & Convert

Generate the SavedModel and convert it to TFLite

3

Verify

Perform evaluation on random data to verify the results

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Getting a basic model running

Get started

1

Build a model

Create a simple model for (y = 2x - 1) from simulated data and train it

2

Export & Convert

Generate the SavedModel and convert it to TFLite

3

Verify

Perform evaluation on random data to verify the results

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Getting a basic model running

Get started

1

Build a model

Create a simple model for (y = 2x - 1) from simulated data and train it

2

Export & Convert

Generate the SavedModel and convert it to TFLite

3

Verify

Perform evaluation on random data to verify the results

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Getting a basic model running

Get started

1

Build a model

Create a simple model for (y = 2x - 1) from simulated data and train it

2

Export & Convert

Generate the SavedModel and convert it to TFLite

3

Verify

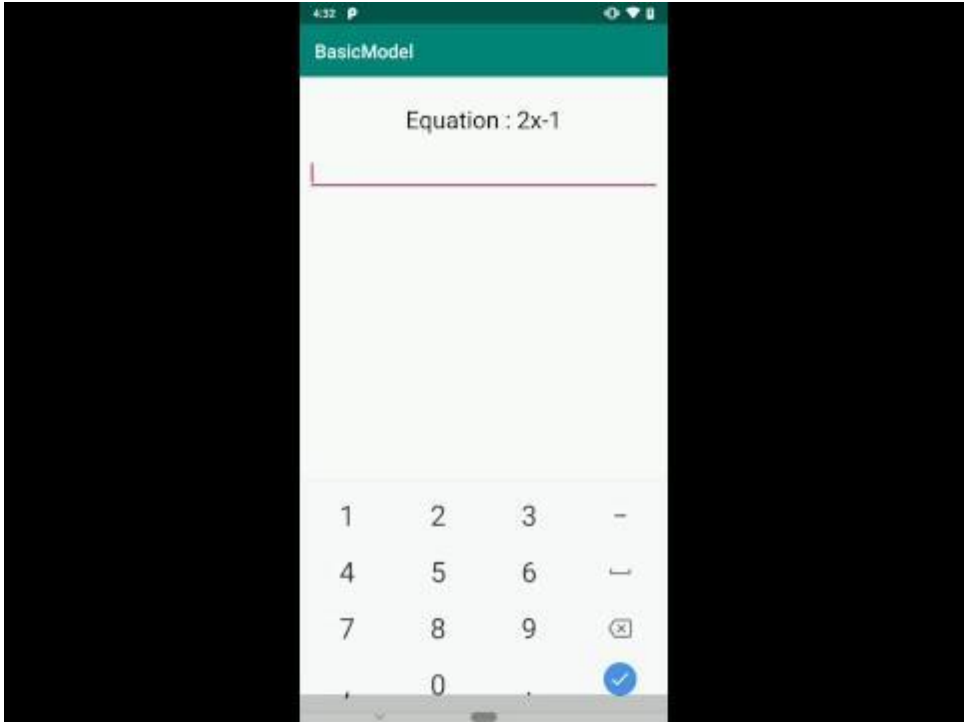
Perform evaluation on random data to verify the results

4

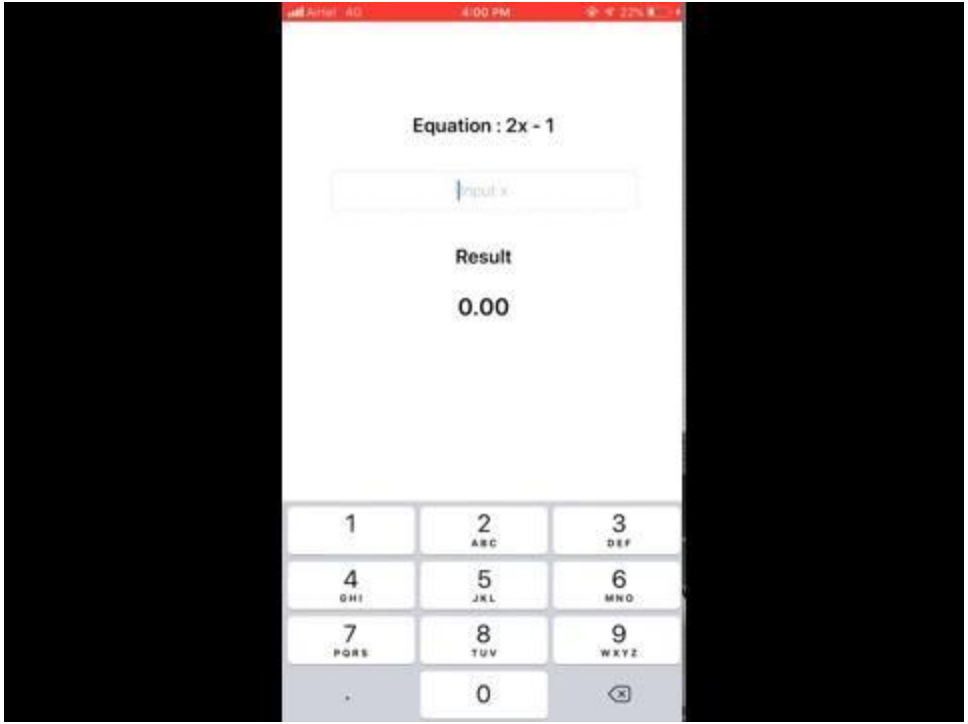
Deploy

Deploy the converted model on a mobile device (Android/iOS)

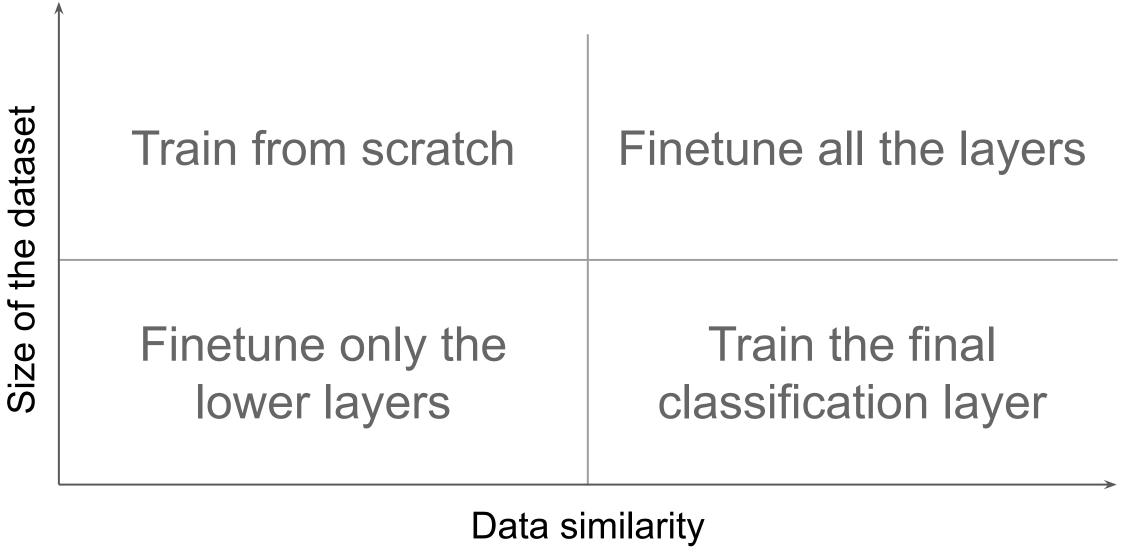
Android

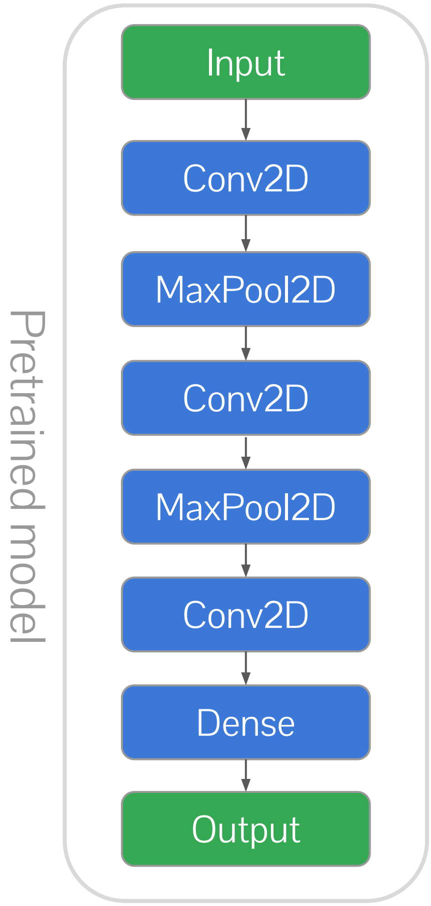


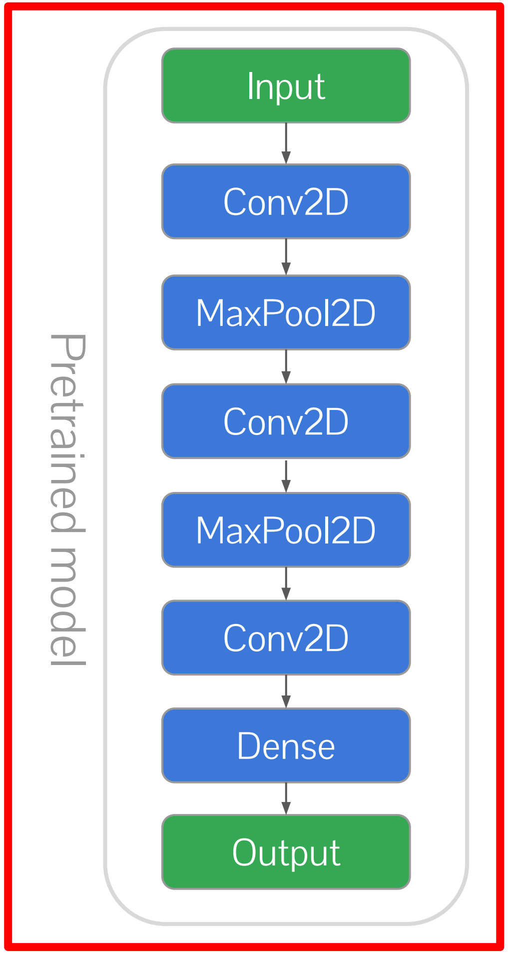
iOS



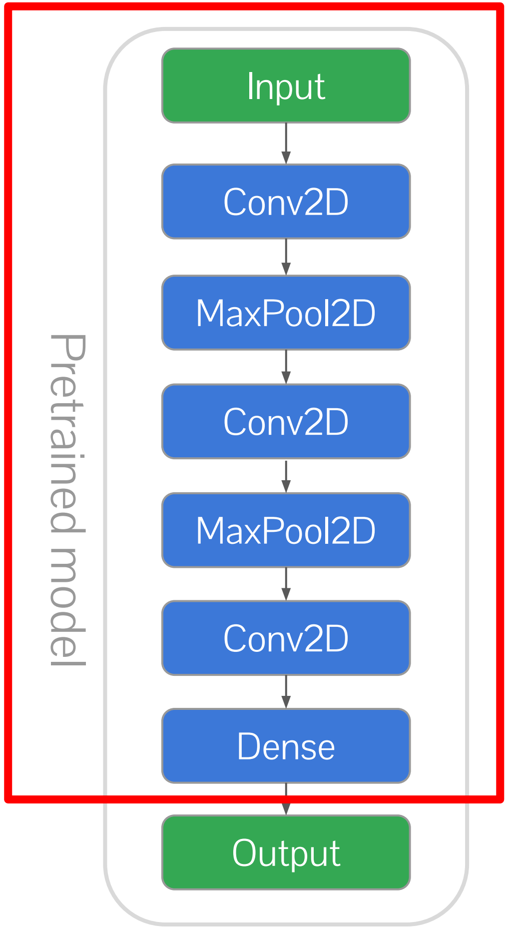
Transfer Learning

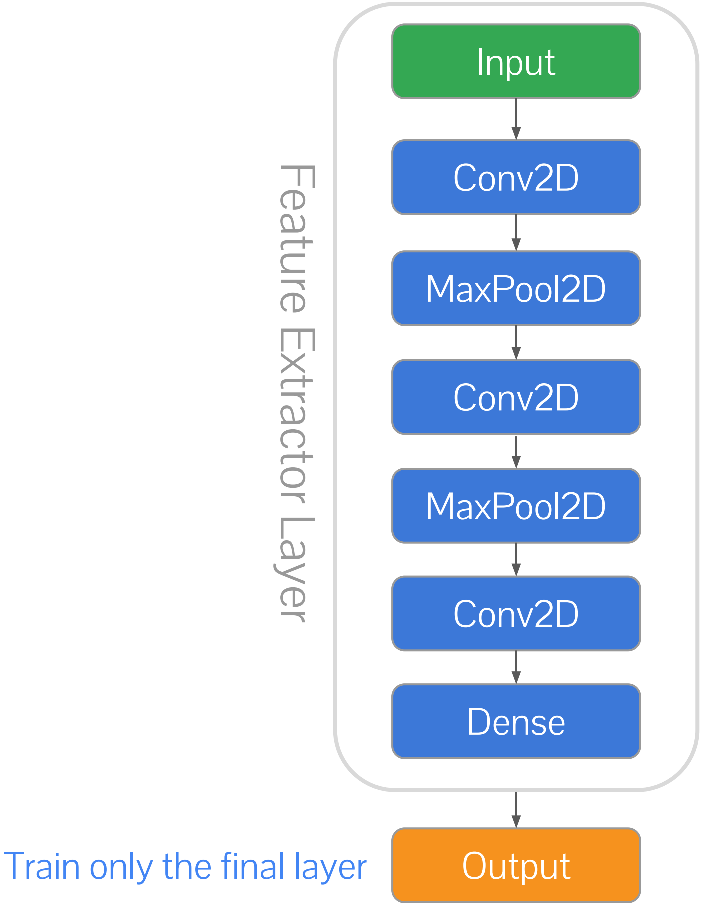



Pretrained model
Input
Conv2D
MaxPool2D
Conv2D
MaxPool2D
Conv2D
Dense
Output



Pretrained model
Input
Conv2D
MaxPool2D
Conv2D
MaxPool2D
Conv2D
Dense
Output


Pretrained model
Input
Conv2D
MaxPool2D
Conv2D
MaxPool2D
Conv2D
Dense
Output


Feature Extractor Layer
Input
Conv2D
MaxPool2D
Conv2D
MaxPool2D
Conv2D
Dense
Output
Train only the final layer

Transfer Learning on Cats vs Dogs with TensorFlow Hub

Get started

1

Prepare the dataset

Download the Cats vs. Dogs tf.Dataset, split into sets (train, val, test), and preprocess

2

Transfer Learning

Choose a feature vector module (MobileNet V2) from TFHub and perform transfer learning

3

Export & Convert

Export the trained model to SavedModel and convert it to TFLite

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Transfer Learning on Cats vs Dogs with TensorFlow Hub

Get started

1

Prepare the dataset

Download the Cats vs. Dogs tf.Dataset, split into sets (train, val, test), and preprocess

2

Transfer Learning

Choose a feature vector module (MobileNet V2) from TFHub and perform transfer learning

3

Export & Convert

Export the trained model to SavedModel and convert it to TFLite

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Transfer Learning on Cats vs Dogs with TensorFlow Hub

Get started

1

Prepare the dataset

Download the Cats vs. Dogs tf.Dataset, split into sets (train, val, test), and preprocess

2

Transfer Learning

Choose a feature vector module (MobileNet V2) from TFHub and perform transfer learning

3

Export & Convert

Export the trained model to SavedModel and convert it to TFLite

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Transfer Learning on Cats vs Dogs with TensorFlow Hub

Get started

1

Prepare the dataset

Download the Cats vs. Dogs tf.Dataset, split into sets (train, val, test), and preprocess

2

Transfer Learning

Choose a feature vector module (MobileNet V2) from TFHub and perform transfer learning

3

Export & Convert

Export the trained model to SavedModel and convert it to TFLite

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)

Transfer Learning on Cats vs Dogs with TensorFlow Hub

Get started

1

Prepare the dataset

Download the Cats vs. Dogs tf.Dataset, split into sets (train, val, test), and preprocess

2

Transfer Learning

Choose a feature vector module (MobileNet V2) from TFHub and perform transfer learning

3

Export & Convert

Export the trained model to SavedModel and convert it to TFLite

4

Deploy

Deploy the converted model on a mobile device (Android/iOS)