Experts

Doing more with TensorFlow Lite

Train, optimize, deploy, and repeat!

Sayak Paul
PyImageSearch
@RisingSayak

Google Developers

Acknowledgement

- The entire PyImageSearch team
- Arun Venkatesan (Google)
- Khanh LeViet (Google)

Ideal audience

- ML Developers having worked on image models (in Keras)
- Mobile Developers looking for ways to plug ML in their applications

Experts

Agenda

- Motivation behind on-device ML
- What is TensorFlow Lite (TF Lite)?
- What can it do?
- Different TF Lite usage scenarios
    - Model optimization
    - Model maker
    - For mobile, embedded, and microcontroller devices
- Some best practices
- QnA

Motivation behind on-device ML

android iOS arm

- Lower latency & close knit interactions
- Network connectivity
- Privacy preserving

What is TensorFlow Lite?

Experts

TensorFlow Lite is a production ready, cross-platform framework for deploying ML on mobile devices and embedded systems

What can it do?

- Optimize your models.

What can it do?

- Optimize your models.
- Take advantage of special hardware accelerators like *Edge TPU* with the use of *delegation*.

What can it do?

- Optimize your models.
- Take advantage of special hardware accelerators like *Edge TPU* with the use of *delegation*.
- Different tools for easy integration of ML in mobile, embedded, and microcontroller-based applications.

What can it do?

- Optimize your models.
- Take advantage of special hardware accelerators like *Edge TPU* with the use of *delegation*.
- Different tools for easy integration of ML in mobile, embedded, and microcontroller-based applications.
- And more: https://www.tensorflow.org/lite

Different TF Lite usage scenarios

- Model optimization
- Model maker
- For mobile, embedded, and microcontroller devices

Model optimization

- Why is it required?

Model optimization

- Why is it required?
  - Size reduction

Model optimization

- Why is it required?
  - Size reduction
  - Latency reduction

Model optimization

- Why is it required?
  - Size reduction
  - Latency reduction
  - Accelerator compatibility

Model optimization

- Why is it required?
- Different optimization options in TensorFlow

Model optimization

- Why is it required?
- Different optimization options in TensorFlow
  - **Quantization**
  - Pruning

Quantization in TF Lite

What is quantization?

- Works by ***reducing the precision*** of the numbers used to represent a model's parameters (float-32 mostly).
- This results in a ***smaller model size*** and ***faster computation***.

Quantization in TF Lite

Types of quantization supported by TF Lite

- **Post-training** quantization
- **Quantization-aware** training

Quantization in TF Lite

Post-training quantization in TF Lite

- Happens ***after*** a model is trained.

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]
```

```
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
```

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')


# Train your model

model.fit(x, y, epochs=50)
```

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')


# Train your model

model.fit(x, y, epochs=50)

# Optimize your model

converter = tf.lite.TFLiteConverter.from_keras_model(model)

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

tflite_model = converter.convert()
```

```python
# Optimize your model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
```

- tf.lite.Optimize.Default
- **tf.lite.Optimize.OPTIMIZE_FOR_SIZE**
- tf.lite.Optimize.OPTIMIZE_FOR_LATENCY

```
# Serialize the TF Lite model
f = open("model.tflite", "wb")
f.write(tflite_model)
f.close
```

Post-training quantization in TF Lite

Different forms of post-training quantization available:

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2-3x speedup, accuracy	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, etc.
Float16 quantization	2x smaller, potential GPU acceleration	CPU/GPU

Check out here: Post-training quantization

Quantization in TF Lite

- **Quantization-aware** training compensates for the information loss introduced by quantization.

Quantization in TF Lite

- **Quantization-aware** training compensates for the information loss introduced for quantization.
- **Quantization-aware** training is possible with the Model Optimization Toolkit. Check out: Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy.

Trade-offs: Speed vs. Accuracy

Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Accuracy loss	CPU
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Smaller accuracy loss	CPU, EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, EdgeTPU, Hexagon DSP

Check out here: [Model optimization]

A closer look at latency

Experts

Incredible Performance

CPU
37 ms

Floating
point

Mobilenet V1

Pixel 4 *- Single Threaded CPU, February 2020*

Incredible Performance

CPU
37 ms

Floating
point

CPU 2.8x
13 ms

Quantized
Fixed-point

Mobilenet V1

Pixel 4 *- Single Threaded CPU, February 2020*

Incredible Performance

CPU 37 ms	CPU 2.8x 13 ms	GPU 6.2x 5 ms
Floating point	Quantized Fixed-point	OpenCL Float16

Mobilenet V1

***Pixel 4** - Single Threaded CPU, February 2020*

Incredible Performance

CPU 37 ms	CPU 2.8x 13 ms	GPU 6.2x 5 ms	EdgeTPU 18.5x 2 ms
Floating point	Quantized Fixed-point	OpenCL Float16	Quantized Fixed-point

Mobilenet V1

***Pixel 4** - Single Threaded CPU, February 2020*

Different TF Lite usage scenarios

- Model optimization
- **Model maker**
- For mobile, embedded, and microcontroller devices

/** Without TensorFlow Lite Model Maker */

1. Load data.

```
IMAGE_SIZE = 224
BATCH_SIZE = 64
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1./255,
   validation_split=0.2)
train_data = datagen.flow_from_directory(
   'flower_photos/',
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='training')
test_data = datagen.flow_from_directory(
   base_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='testing')
```

/** Without TensorFlow Lite Model Maker */

1. Load data.

```
IMAGE_SIZE = 224
BATCH_SIZE = 64
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1./255,
   validation_split=0.2)
train_data = datagen.flow_from_directory(
   'flower_photos/',
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='training')
test_data = datagen.flow_from_directory(
   base_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='testing')
```

2. Create and retrain the model.

```
# Download the headless model
feature_extractor_url =
  "https://tfhub.dev/google/efficientnet/b0/feature-vector/1"
feature_extractor_layer = hub.KerasLayer(feature_extractor_url,
                                   input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
feature_extractor_layer.trainable = False

# Attach a classification head
model = tf.keras.Sequential([
  feature_extractor_layer,
  layers.Dense(train_data.num_classes, activation='softmax')
])
```

2. Create and retrain the model. (continued)

```
# train the model
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss='categorical_crossentropy',
  metrics=['acc'])
steps_per_epoch = np.ceil(train_data.samples/train_data.batch_size)
history = model.fit(train_data, epochs=2,
                    steps_per_epoch=steps_per_epoch)
```

/** Without TensorFlow Lite Model Maker */

1. Load data.

```
IMAGE_SIZE = 224
BATCH_SIZE = 64
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1./255,
   validation_split=0.2)
train_data = datagen.flow_from_directory(
   'flower_photos/',
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='training')
test_data = datagen.flow_from_directory(
   base_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='testing')
```

2. Create and retrain the model.

```
# Download the headless model
feature_extractor_url =
  "https://tfhub.dev/google/efficientnet/b0/feature-vector/1"
feature_extractor_layer = hub.KerasLayer(feature_extractor_url,
                                   input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
feature_extractor_layer.trainable = False

# Attach a classification head
model = tf.keras.Sequential([
  feature_extractor_layer,
  layers.Dense(train_data.num_classes, activation='softmax')
])
```

2. Create and retrain the model. (continued)

```
# train the model
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss='categorical_crossentropy',
  metrics=['acc'])
steps_per_epoch = np.ceil(train_data.samples/train_data.batch_size)
history = model.fit(train_data, epochs=2,
                    steps_per_epoch=steps_per_epoch)
```

3. Predict and evaluate results.

```
class_names = sorted(test_data.class_indices.items(),
                     key=lambda pair:pair[1])
class_names = np.array([key.title() for key, value in class_names])
for image_batch, label_batch in test_data:
  predicted_batch = model.predict(image_batch)
  predicted_id = np.argmax(predicted_batch, axis=-1)
  predicted_label_batch = class_names[predicted_id]
model.evaluate(test_data)
```

/** Without TensorFlow Lite Model Maker */

1. Load data.

```
IMAGE_SIZE = 224
BATCH_SIZE = 64
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1./255,
   validation_split=0.2)
train_data = datagen.flow_from_directory(
   'flower_photos/',
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='training')
test_data = datagen.flow_from_directory(
   base_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   batch_size=BATCH_SIZE,
   subset='testing')
```

2. Create and retrain the model.

```
# Download the headless model
feature_extractor_url =
  "https://tfhub.dev/google/efficientnet/b0/feature-vector/1"
feature_extractor_layer = hub.KerasLayer(feature_extractor_url,
                                   input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
feature_extractor_layer.trainable = False

# Attach a classification head
model = tf.keras.Sequential([
  feature_extractor_layer,
  layers.Dense(train_data.num_classes, activation='softmax')
])
```

2. Create and retrain the model. (continued)

```
# train the model
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss='categorical_crossentropy',
  metrics=['acc'])
steps_per_epoch = np.ceil(train_data.samples/train_data.batch_size)
history = model.fit(train_data, epochs=2,
                    steps_per_epoch=steps_per_epoch)
```

3. Predict and evaluate results.

```
class_names = sorted(test_data.class_indices.items(),
                     key=lambda pair:pair[1])
class_names = np.array([key.title() for key, value in class_names])
for image_batch, label_batch in test_data:
  predicted_batch = model.predict(image_batch)
  predicted_id = np.argmax(predicted_batch, axis=-1)
  predicted_label_batch = class_names[predicted_id]
model.evaluate(test_data)
```

4. Export to TF Lite

```
converter = tf.lite.TF LiteConverter.from_keras_model(model)
TF Lite_model = converter.convert()
with open('flower.TF Lite', 'w') as f:
  f.write(TF Lite_model)
```

TF Lite Model Maker

- A transfer learning library for TF Lite.
- A new Python library lets you customize models for your dataset, without requiring ML expertise.

TF Lite Model Maker

```
# 1. Load data.
data = ImageClassifierDataLoader.from_folder('flower_photos/')

# 2. Customize the model.
model = image_classifier.create(data) # Default model is EfficientNet-Lite0

# 3. Evaluate the model.
loss, accuracy = model.evaluate()

# 4. Export to TF Lite.
model.export('flower_classifier.TF Lite')
```

Check out here: [examples/image_classification.ipynb]()

Model Maker works with

Image

Classification
(MobileNet,
EfficientNet-Lite,
ResNet...**)**

Object detection*

Text

Classification
(BERT
ALBERT-Lite*
MobileBERT*)

QA*

*coming soon**

Source: [Easy on-device ML from prototype to product]()

Where to find pre-trained TF Lite models?

- **TensorFlow Hub (TFHub):** https://tfhub.dev/s?deployment-format=lite&publisher=tensorflow&q=lite
- **Official TF Lite models:** Hosted models

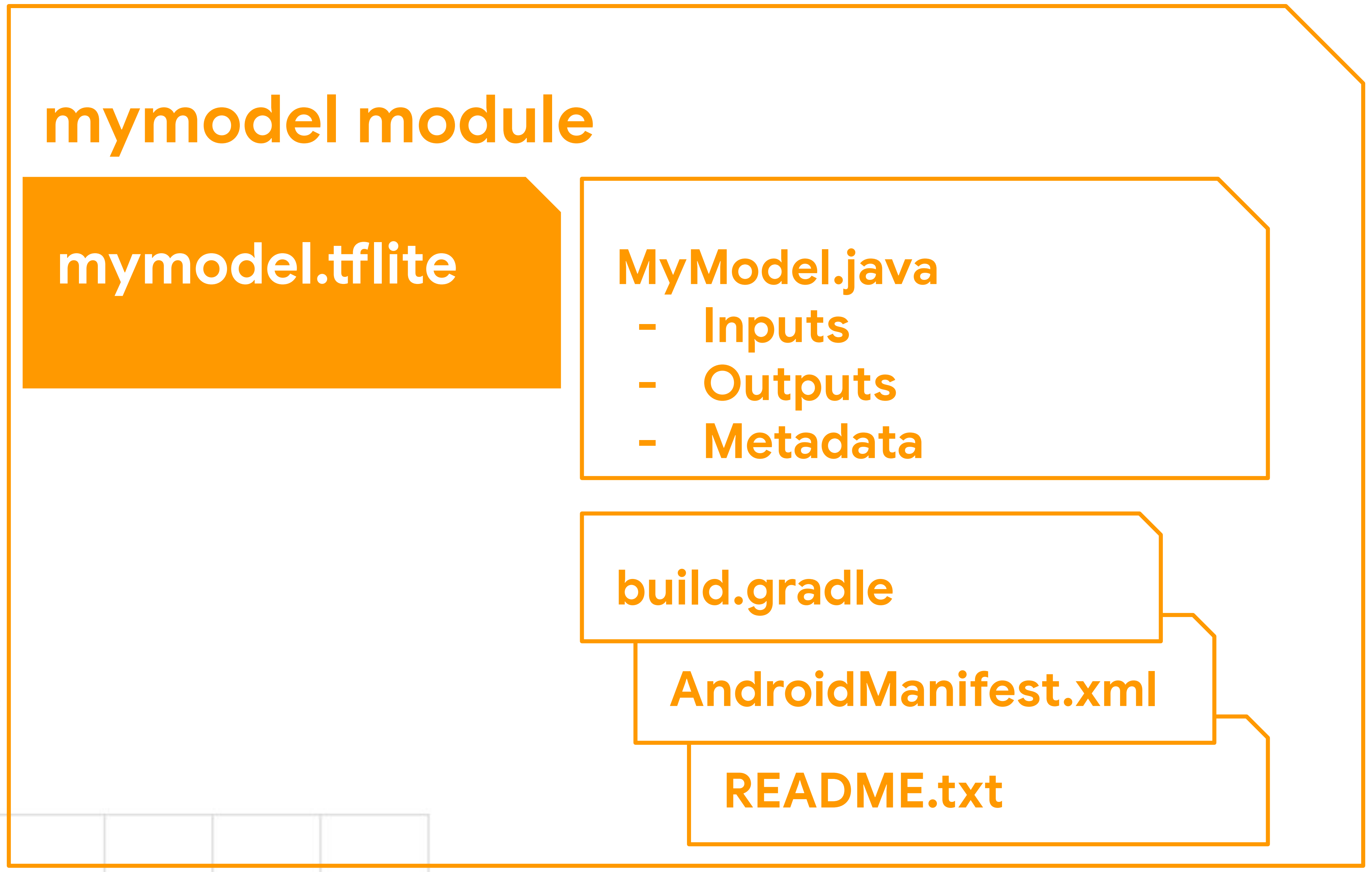
Different TF Lite usage scenarios

- Model optimization
- Model maker
- **For mobile, embedded, and microcontroller devices**

Experts

TF Lite Codegen

Codegen tool ***generates*** an Android wrapper around a TF Lite model and makes it easy to consume!



A command line codegen tool for Android

```
tflite_codegen \
    --model=mobilenet_v1_1.0_224_quant.tflite \
    --package_name="org.tensorflow.lite.myimageclassifier" \
    --model_class_name=MyImageClassifier \
    --destination=./MyImageClassifier
```

Check out: [Generate code from TensorFlow Lite metadata]()

Using it in Android code

Experts

```
/** With TensorFlow Lite codegen */

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

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

// 3. Run inference.
MyImageClassifier.Outputs outputs = classifier.run(inputs);

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

5 lines!!

Prototype

Jump start with example apps

Aa

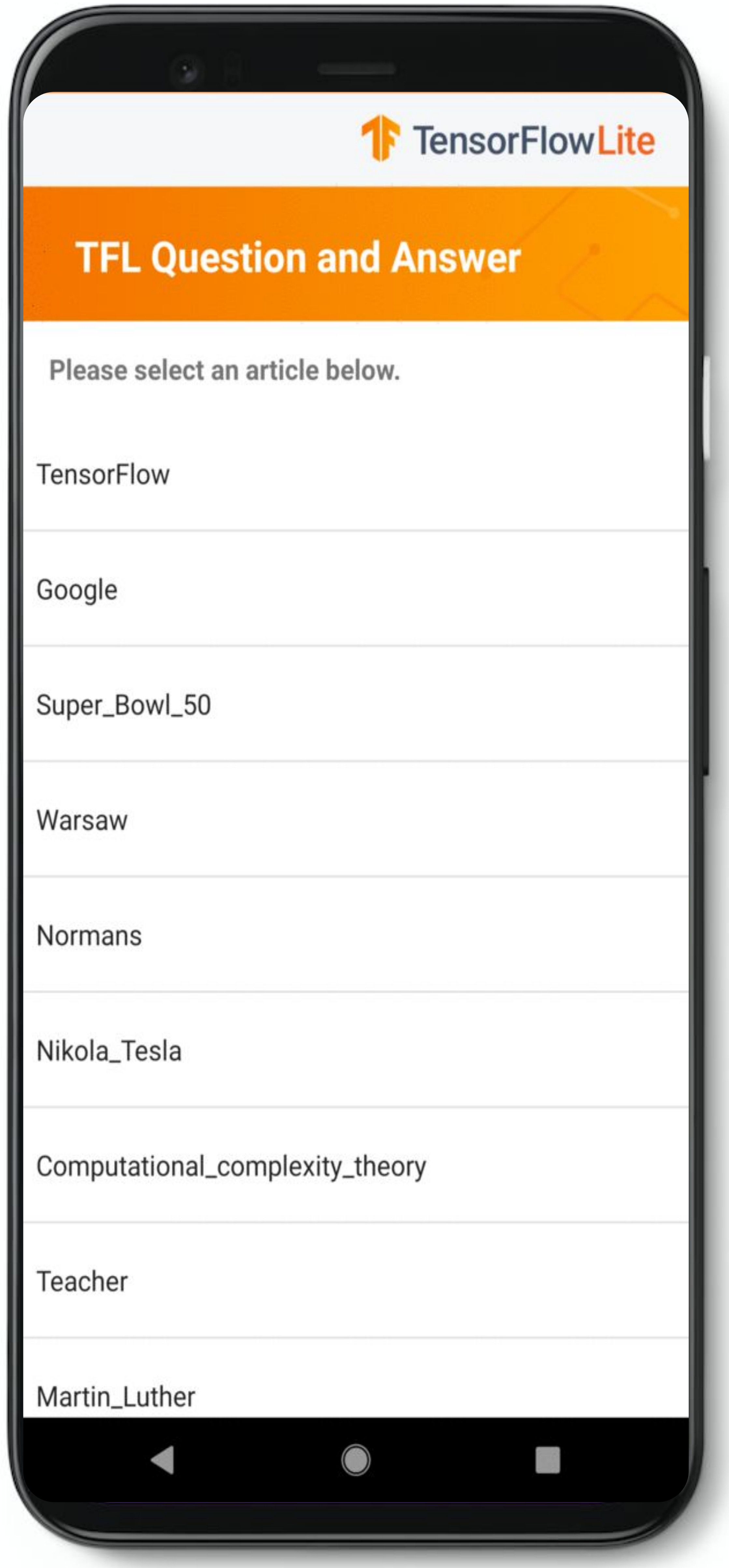
Text

Audio

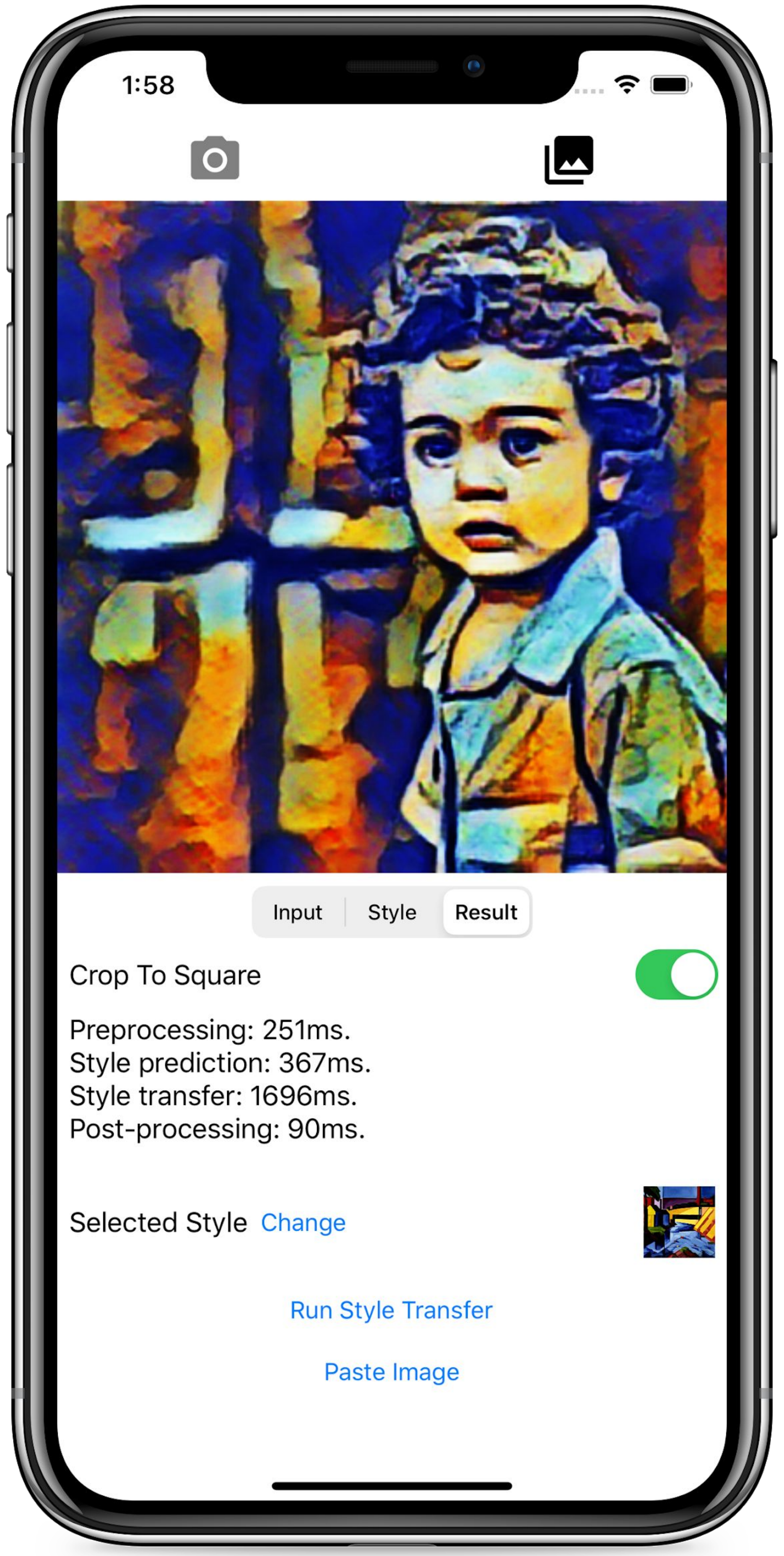
Image

Content

tensorflow.org/lite/examples



Question & Answering
Text



Style Transfer
Content

Folks, hacking with embedded stuff & microcontroller stuff

- Same tooling and framework as TF Lite.
- Developers no longer have to manually build models.
- Hardware level optimizations ***done*** for you.

Folks, hacking with embedded stuff & microcontroller stuff

- Build TensorFlow Lite for ARM64 boards
- Build TensorFlow Lite for Raspberry Pi

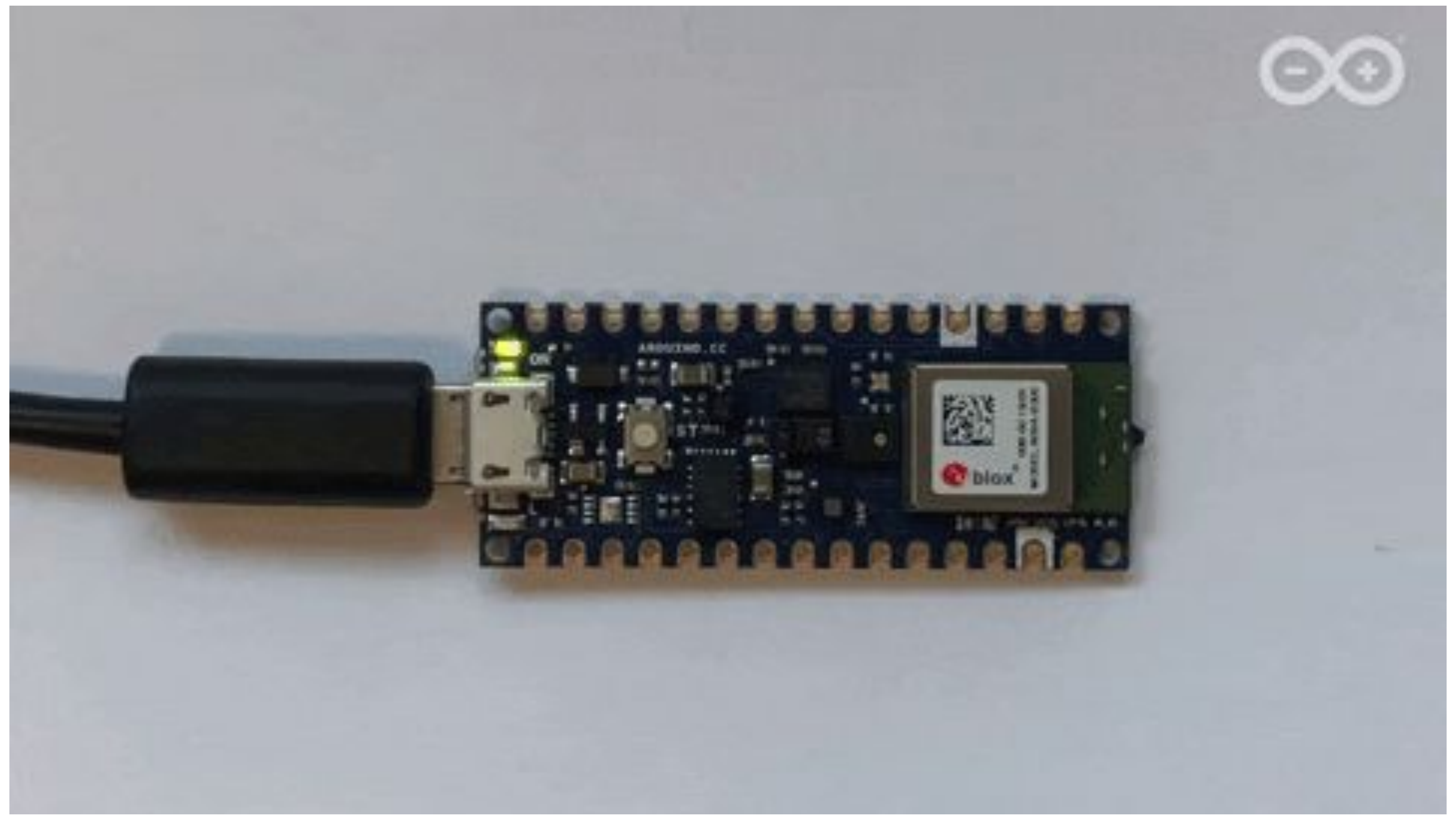
Folks, hacking with embedded stuff & microcontroller stuff

- Tremendous speed up with Edge TPU compatible TF Lite models

Check out here: [Edge TPU performance benchmarks | Coral]

Folks, hacking with embedded stuff & microcontroller stuff

- **Launch of official Arduino library** - run example code directly from desktop and web IDEs onto Arduino hardware
- **Speech detection in 5 minutes** - open source models available to get started quickly on Arduino



ARDUINO

https://www.tensorflow.org/lite/microcontrollers

Some TF Lite best practices

Experts

Consider hosted models first

- See if a pre-trained TF Lite model can do the job.

Consider hosted models first

- See if a pre-trained TF Lite models can do the job.
- Different SoTA models available for different domains and tasks.

Prototype

Pre-trained models for all domains

Text

BERT
ALBERT
MobileBERT
DistilBERT*
SmartReply

Image

EfficientNet-Lite
PoseNet v2
Magenta
DeepLab V3
SSD-MobileNet
MNasNet
MobileNet

Audio

Speech Commands
DeepSpeech*

Content

Style Transfer

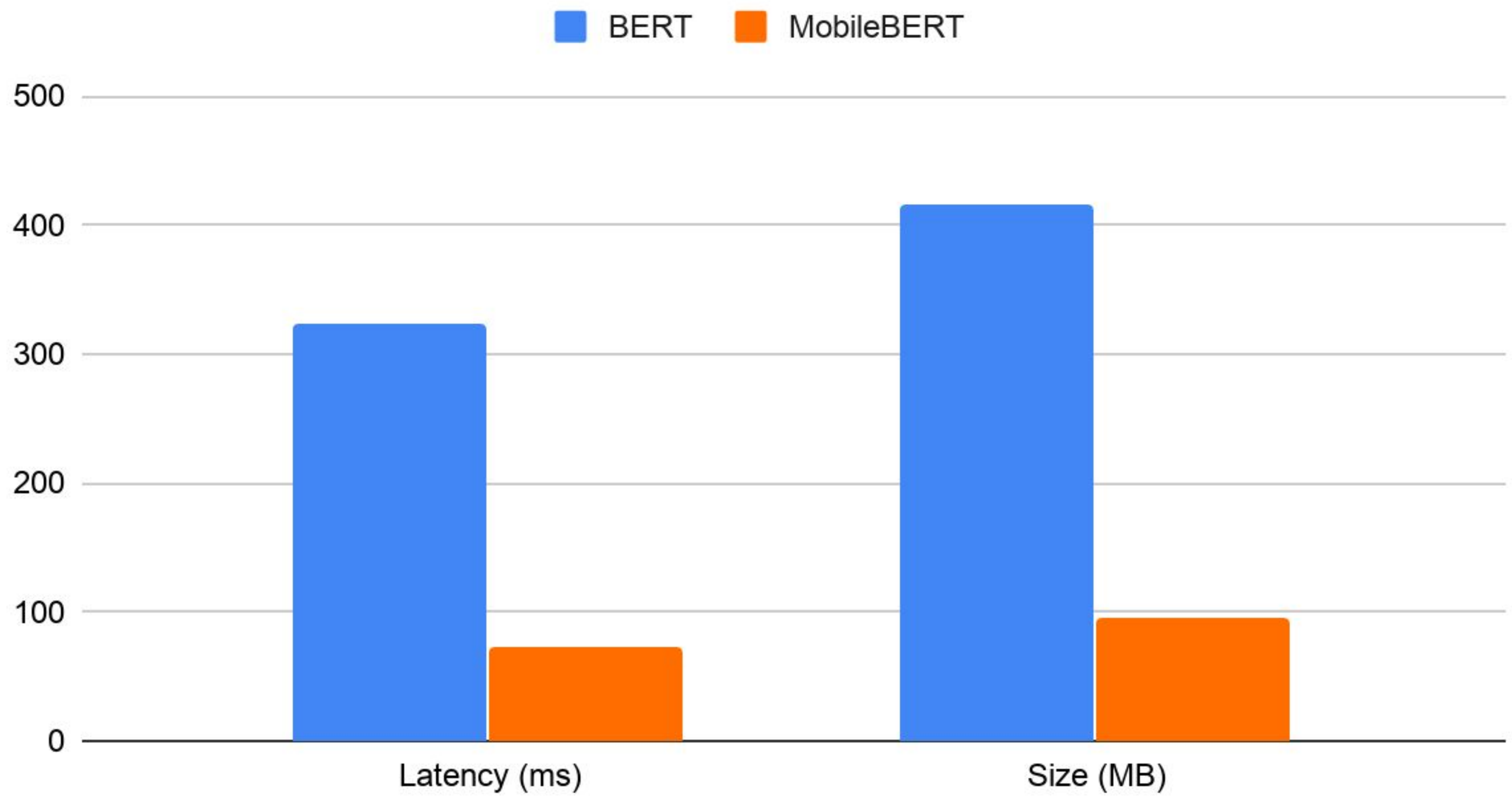
Available now on TensorFlow Hub and GitHub*
(tfhub.dev, tensorflow.org/lite/models,
github.com/margaretmz/awesome-tflite)

Prototype

State of the Art NLP for Mobile

MobileBERT and ALBERT

- **Faster** and **smaller** than BERT
- Even works for low-tier CPU
- 4.4x speedup (74 ms)
- 4x size reduction (< 100 MB)
- Same accuracy



Pixel 4 *- CPU, 4 Threads, Sequence length 128 , Vocab size 30K, October 2019*

* ***ALBERT-Lite*** *available in TFHub*

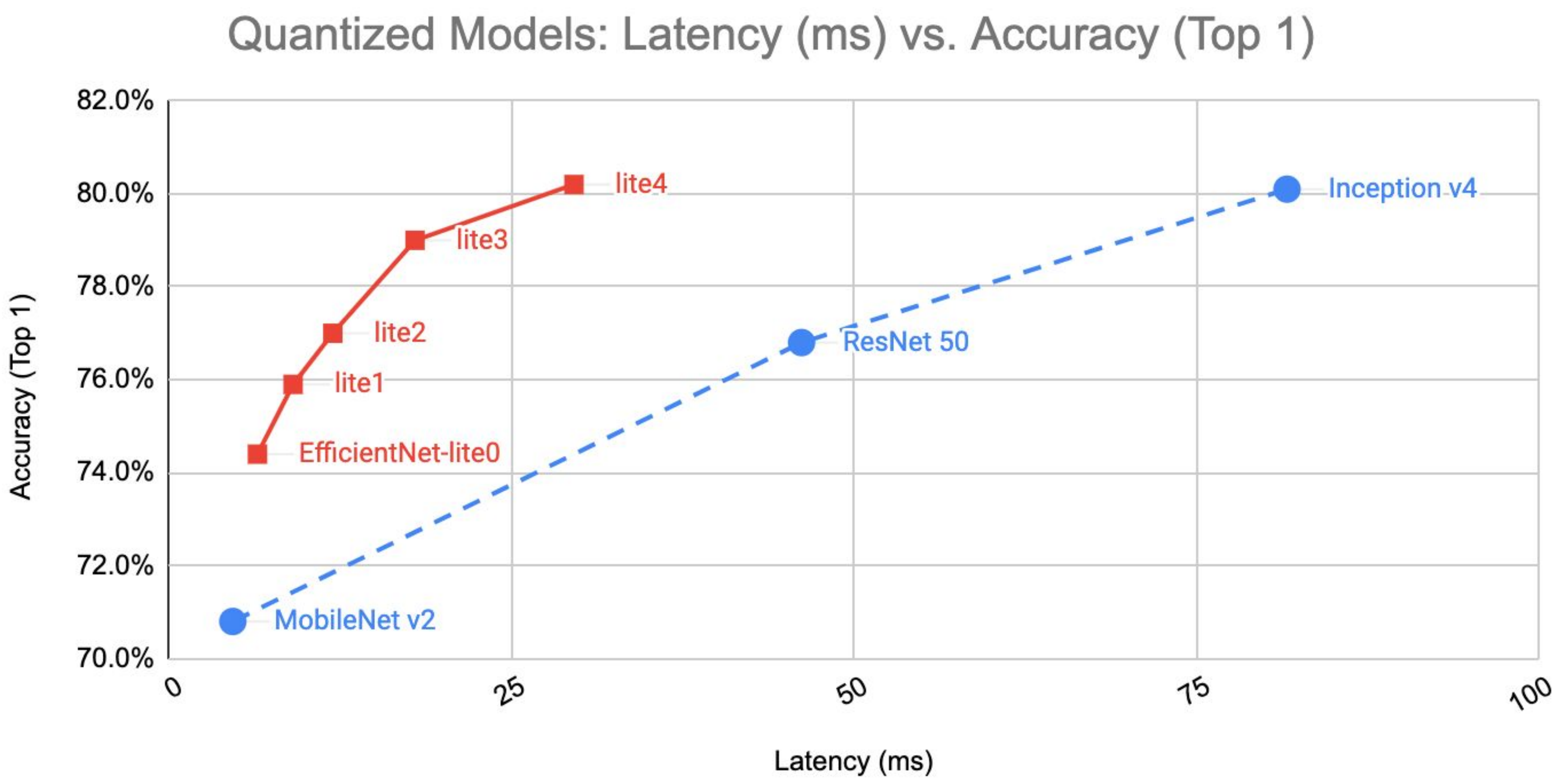
* ***Quantized MobileBERT*** *coming soon*

Prototype

State of the Art Vision for Mobile

EfficientNet-Lite

- SOTA Vision model for image classification
  - Higher accuracy with similar model size and latency
  - E.g. lite4 with 80.4% top-1 accuracy and 30ms on CPU
- Multiple variants for your need, from low latency and model size to high accuracy model



Pixel 4 *- CPU, 4 Threads, March 2020*

Know your trade-offs and optimize accordingly

- Is accuracy super important for your application?

Know your trade-offs and optimize accordingly

- Is accuracy super important for your application?
- Or can it be compensated with speed?

Know your trade-offs and optimize accordingly

- Is accuracy super important for your application?
- Or can it be compensated with speed?
- Or would you want to have a balance between the two?

Know your trade-offs and optimize accordingly

Refer this chart and figure out what works best for you -

Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Accuracy loss	CPU
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Smaller accuracy loss	CPU, EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, EdgeTPU, Hexagon DSP

Source: [Model optimization]

Use delegates whenever possible

"A TensorFlow Lite delegate is a way to delegate part or all of graph execution to another executor."

- TensorFlow Lite delegates

Use delegates whenever possible

Different delegates available in TF Lite:

- GPU (Cross-platform, Float32 & Float16)
- TPU (Edge TPU, Int8)
- NNAPI for newer Android devices
- Hexagon for older Android devices
- Core ML for newer iPhones and iPads

Use delegates whenever possible

Different delegates available in TF Lite:

- GPU (Cross-platform, Float32 & Float16)
- TPU (Edge TPU, Int8)
- NNAPI for newer Android devices
- Hexagon for older Android devices
- Core ML for newer iPhones and iPads

Check out the guide on delegates:
TensorFlow Lite delegates

Know about the support for target device

- In case of optimizing custom models, know which layers are supported.

Know about the support for target device

- In case of optimizing custom models, know which layers are supported.
- In which precision are they supported?
  - Float-16
  - Int8
  - Hybrid

Know about the support for target device

- In case of optimizing custom models, know which layers are supported.
- In which precision are they supported?
- Your target device might not support Float16 (Edge TPU).

Know more here -

[Performance best practices]

Experts

Find out more

- TensorFlow Lite guide
- TensorFlow Lite: ML for mobile and IoT devices (TF Dev Summit '20)
- Easy on-device ML from prototype to product
- How TensorFlow Lite helps you from prototype to product
- Introduction to TensorFlow Lite
- Device-based Models with TensorFlow Lite

TF Lite team needs your help

- Contribute to the ongoing list of examples
- Provide with feedback to the team
- Come up with your own ideas

Fill out the developer survey here: bit.ly/tfl-survey
Questions? tflite@tensorflow.org

Slides available here -

https://bit.ly/tfl-pune

Experts

Experts

Thank You!

Sayak Paul
PyImageSearch
@RisingSayak