Mobile Deep Inference

Sam Ogden - CS4518 Oct. 12th, 2021

WPI

Overview

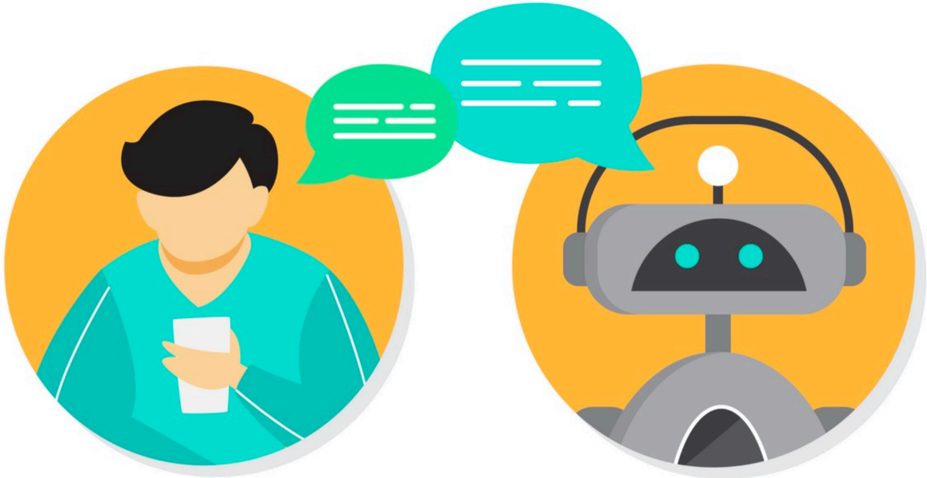
What's to come

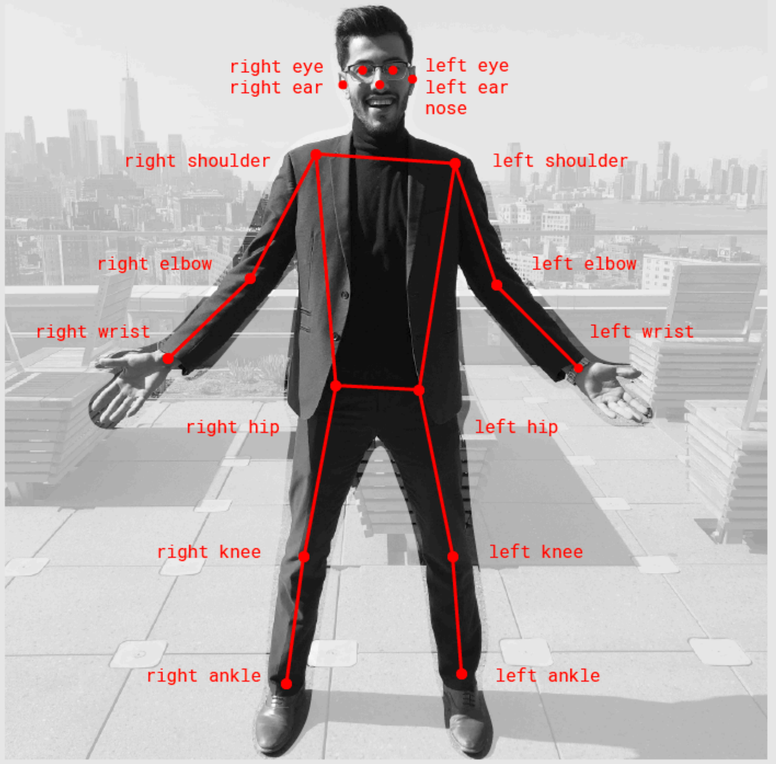
- What is it?
- Why is it challenging?
- How do researchers make it work?
- How do *we* make it work?

What is it?

What is it?

Deep Learning Inference

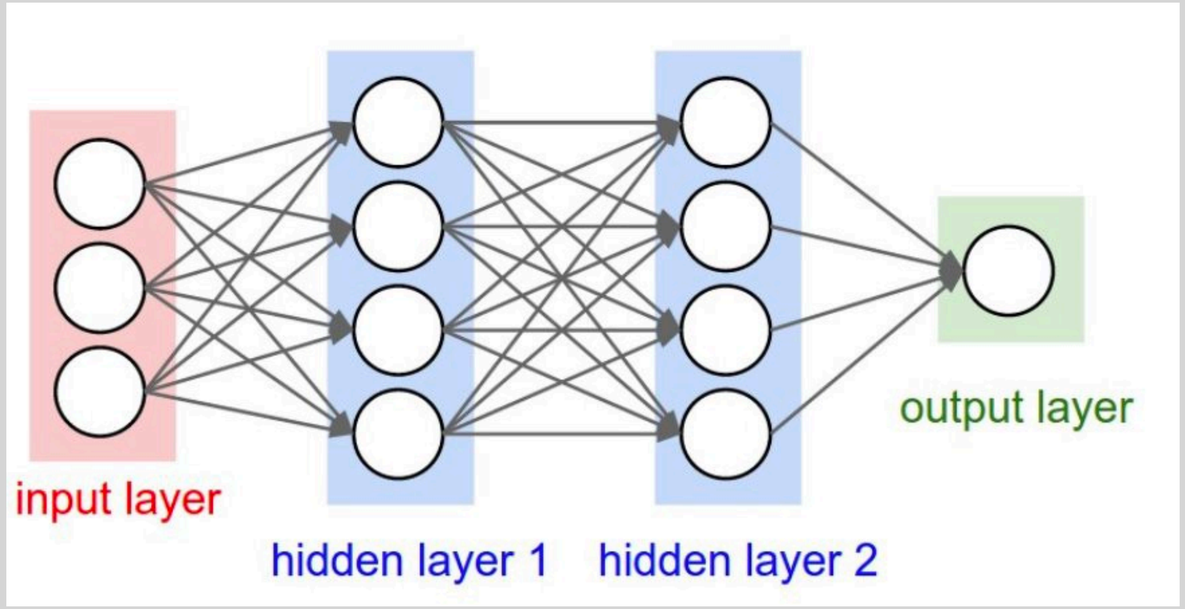




17 Pose Keypoints
Returned by PoseNet

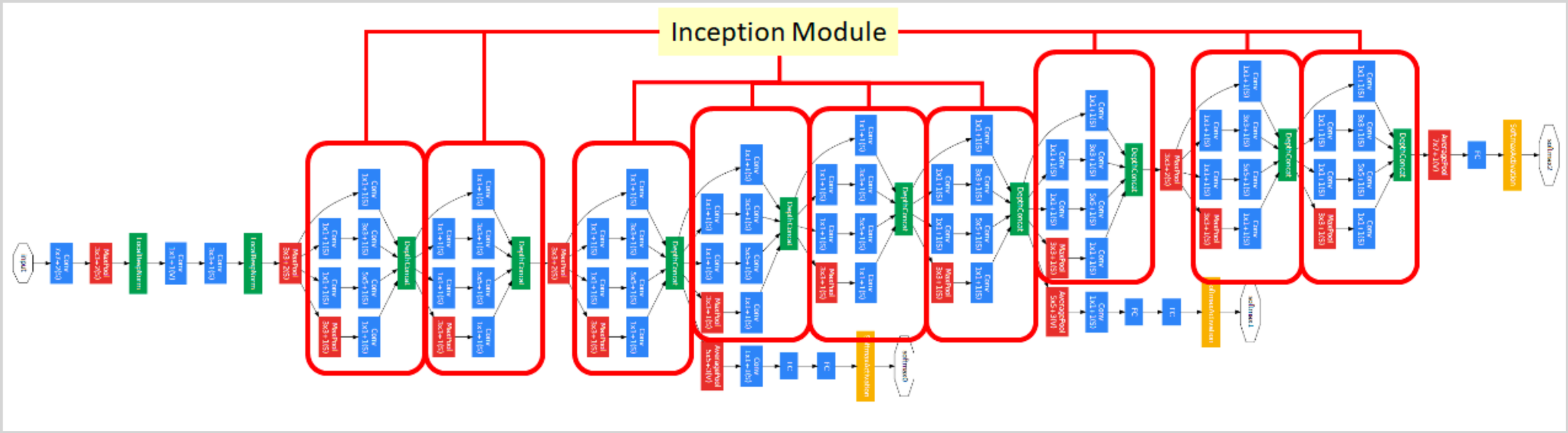
What is it?

Deep Learning Models



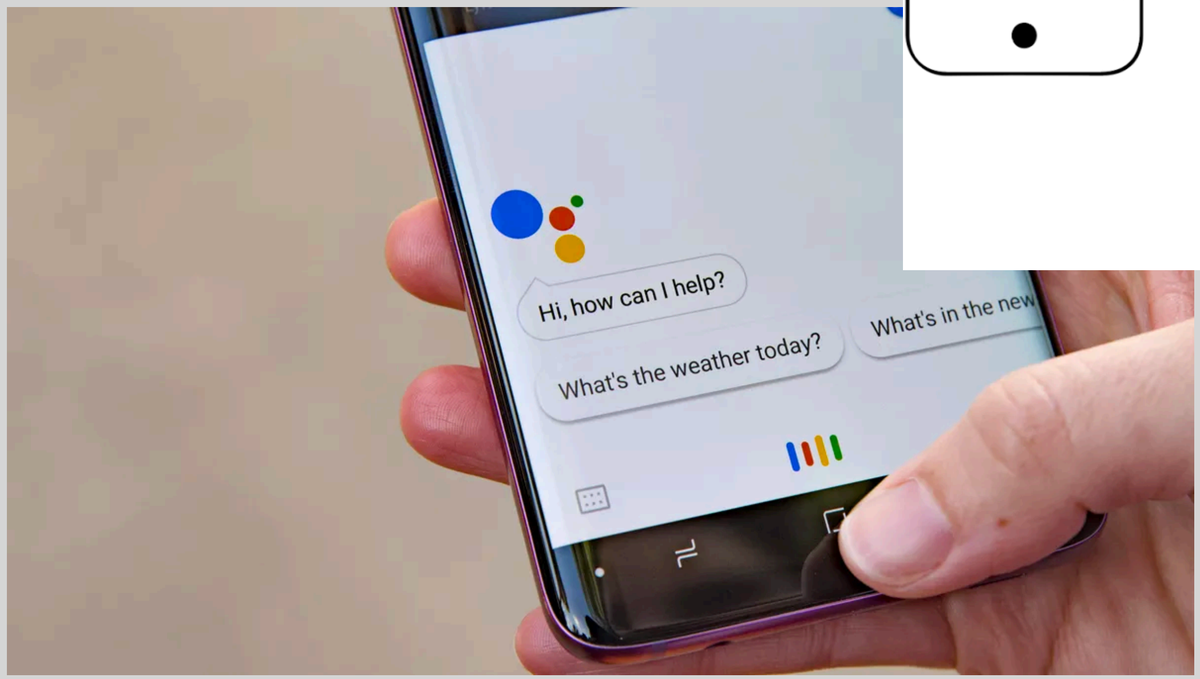
What is it?

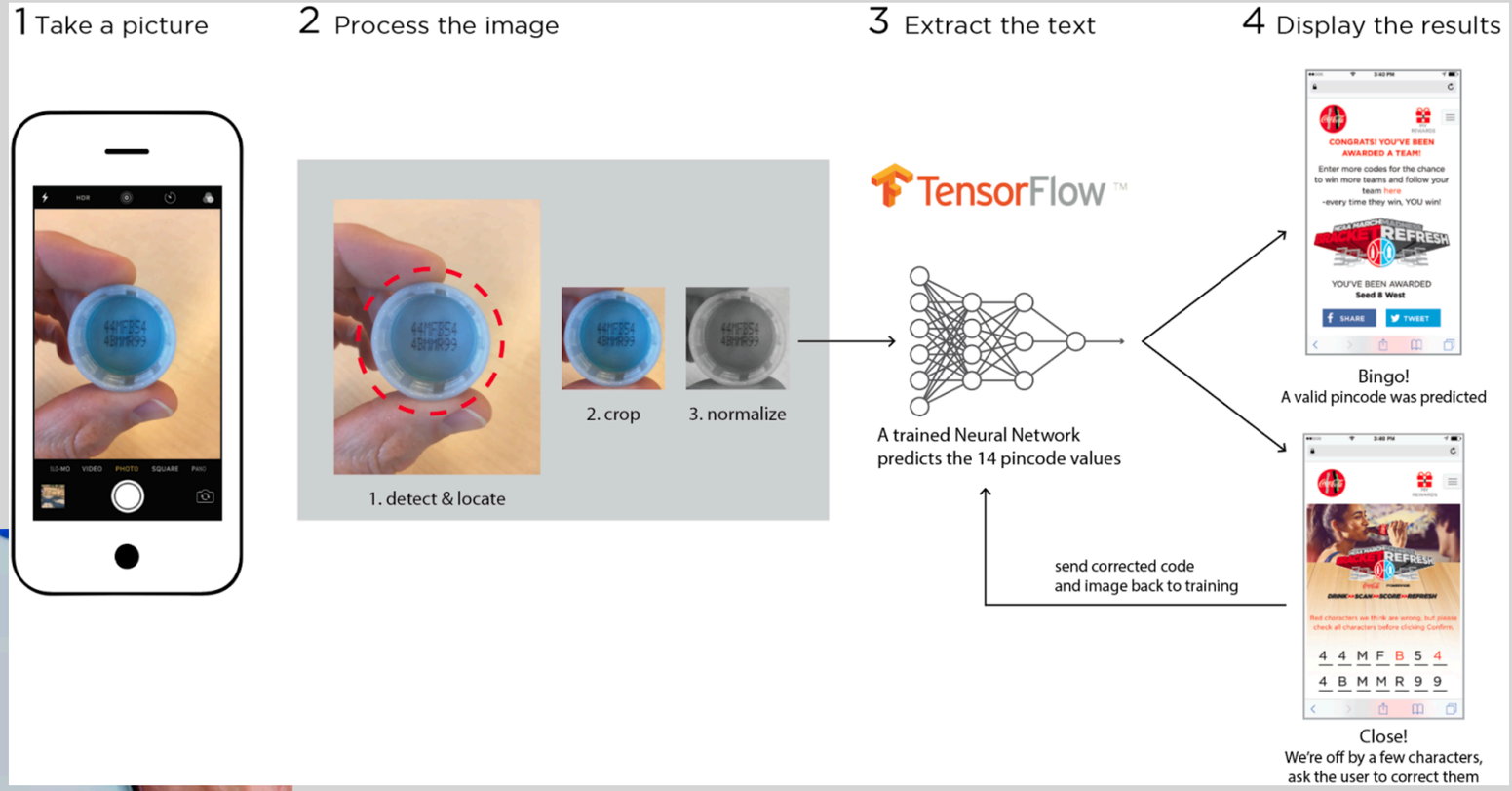
Deep Learning Models



What is it?

Mobile Deep Inference





- https://developers.googleblog.com/2017/09/how-machine-learning-with-tensorflow.html

Why is it challenging?

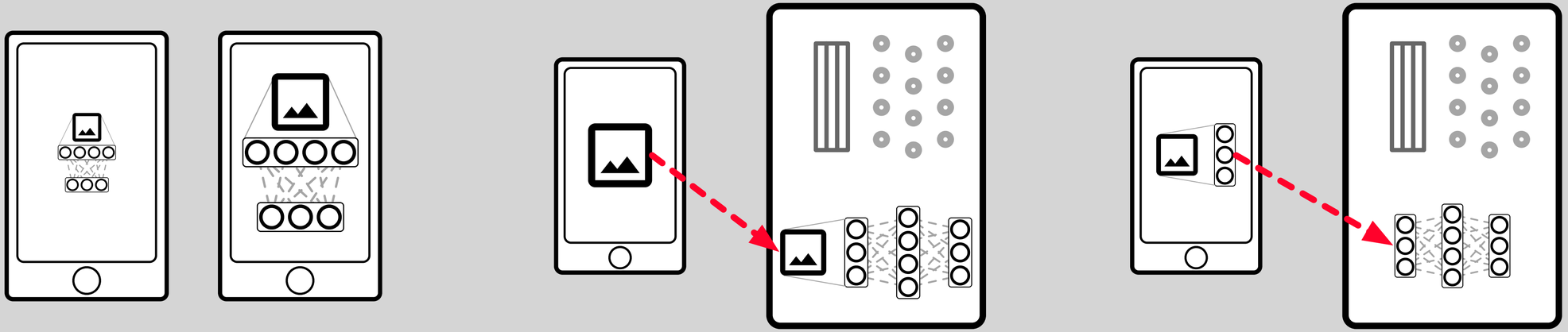
Why is it challenging?

Key Challenges Overview

- Large model size
- Constrained mobile devices
- Lots of different applications
- Latency, latency, latency

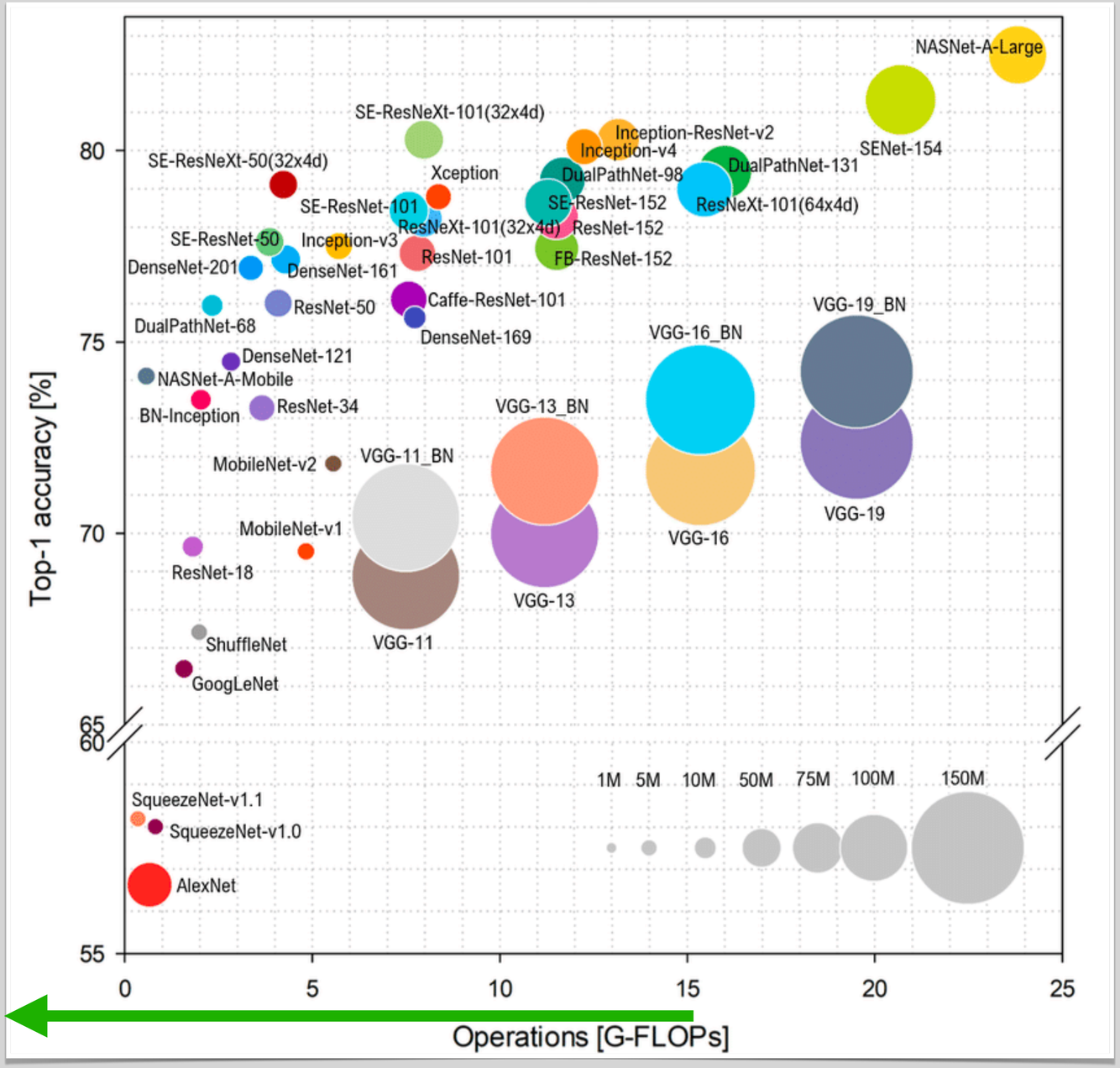
Why is it challenging?

Three main ways to run



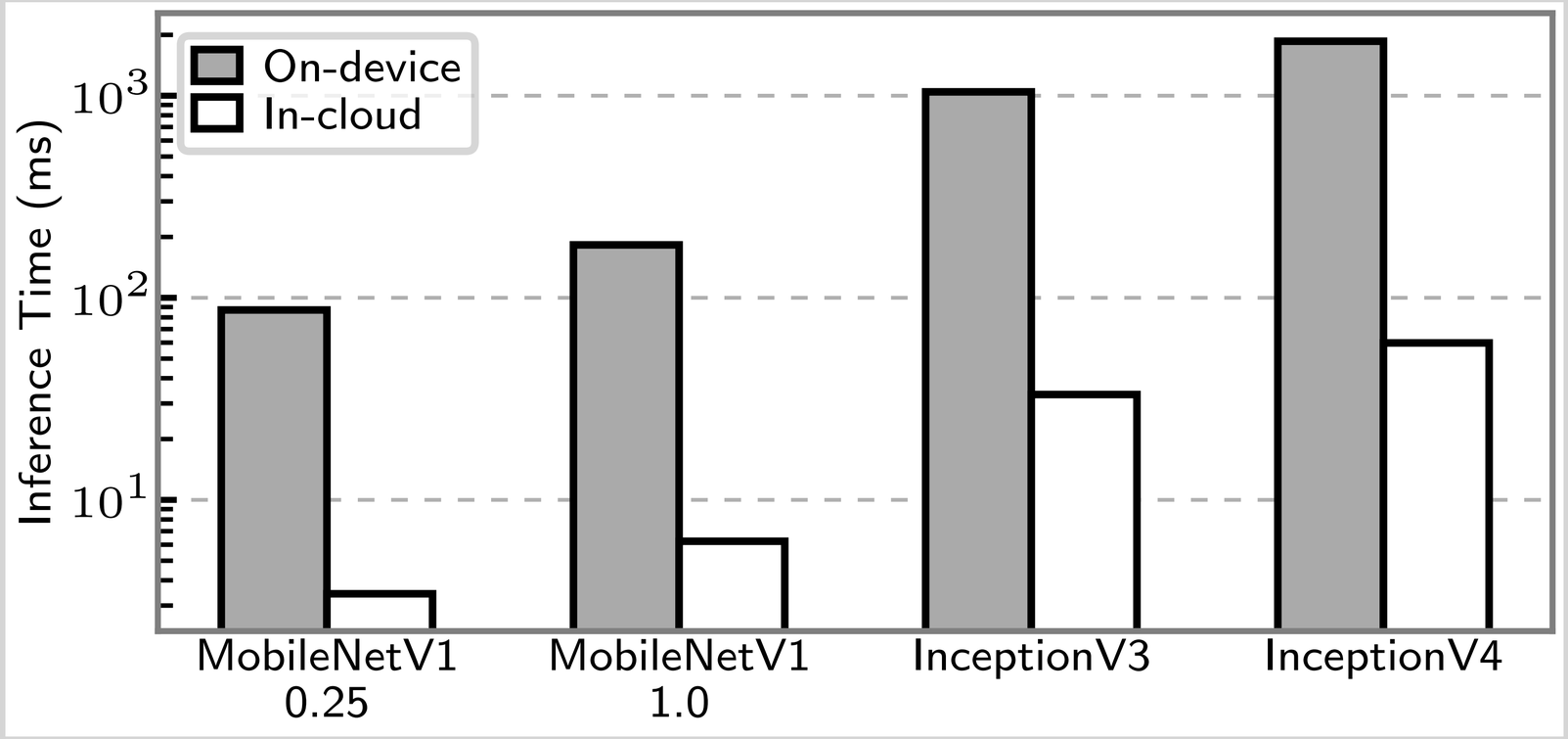
Why is it challenging?

Model Size



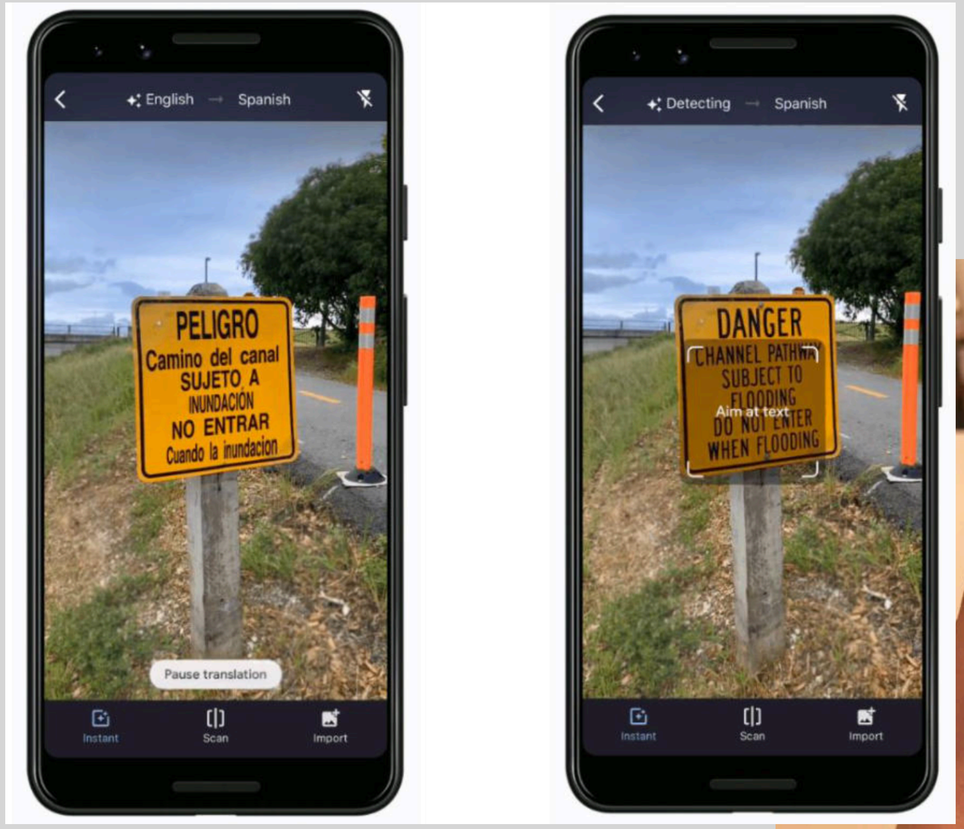
Why is it challenging?

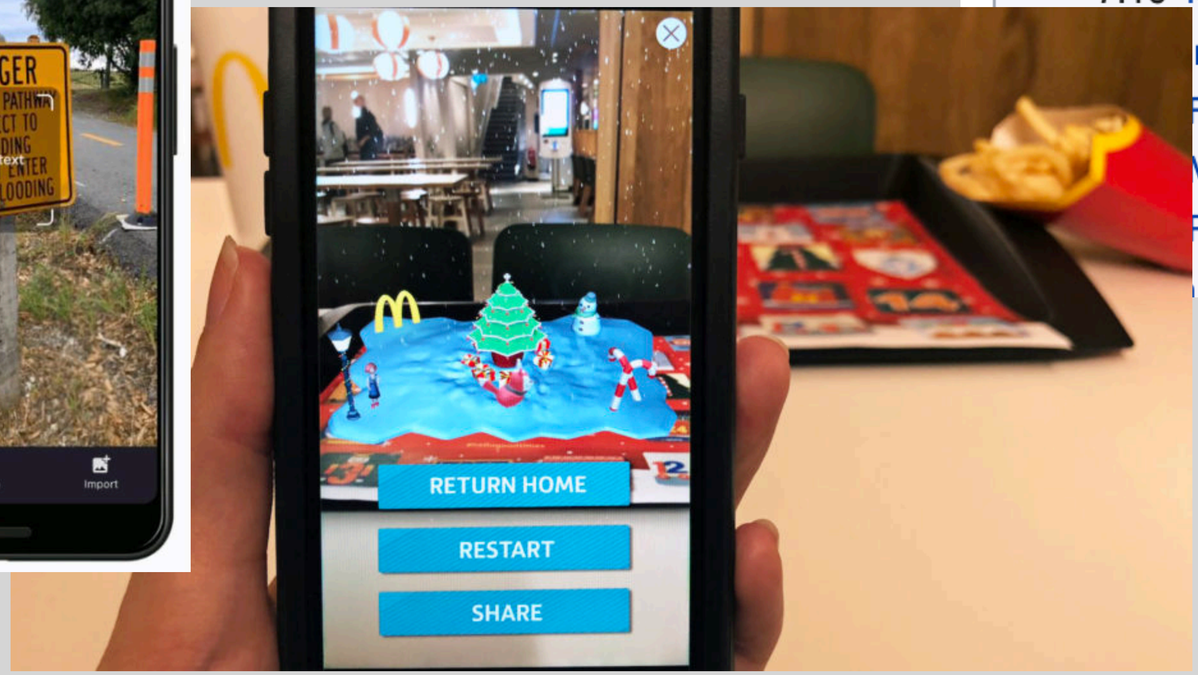
Constrained Mobile Resources



Why is it challenging?

Many different applications

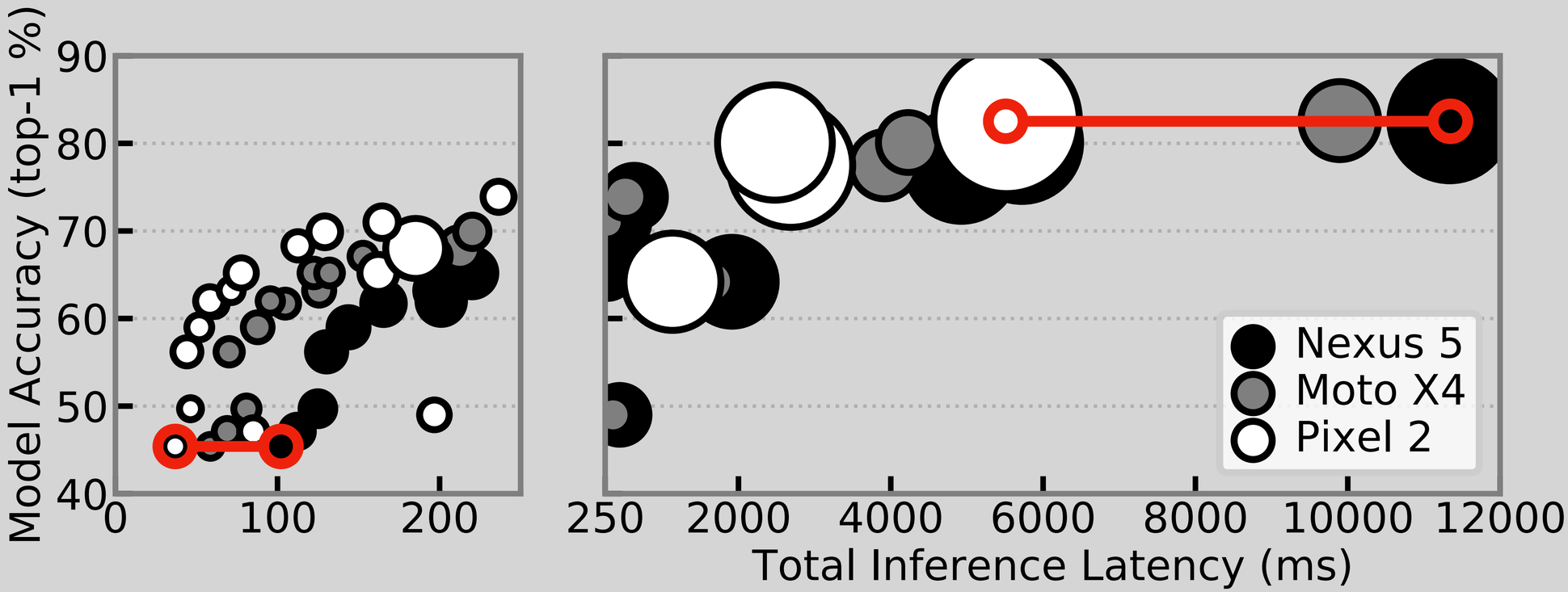




7 Applications
- 7.1 Automatic speech recognition
- 7.2 Image recognition
- 7.3 Visual art processing
- 7.4 Natural language processing
- 7.5 Drug discovery and toxicology
- 7.6 Customer relationship management
- 7.7 Recommendation systems
- 7.8 Bioinformatics
- 7.9 Medical image analysis
- 7.10 Mobile advertising
- mage restoration
- Financial fraud detection
- Military
- Partial differential equations
- to human cognitive and brain devel

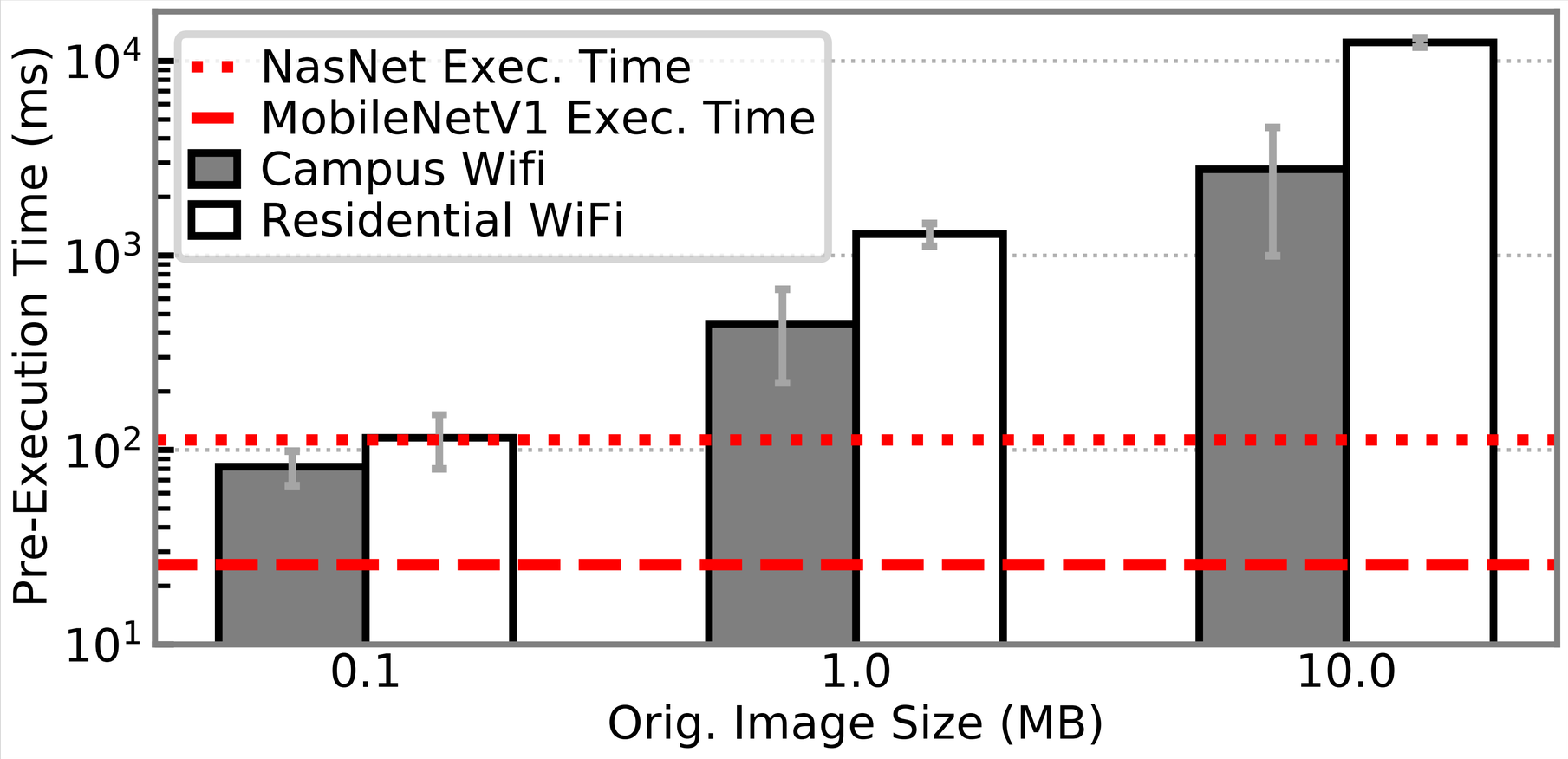
Why is it challenging?

Latency



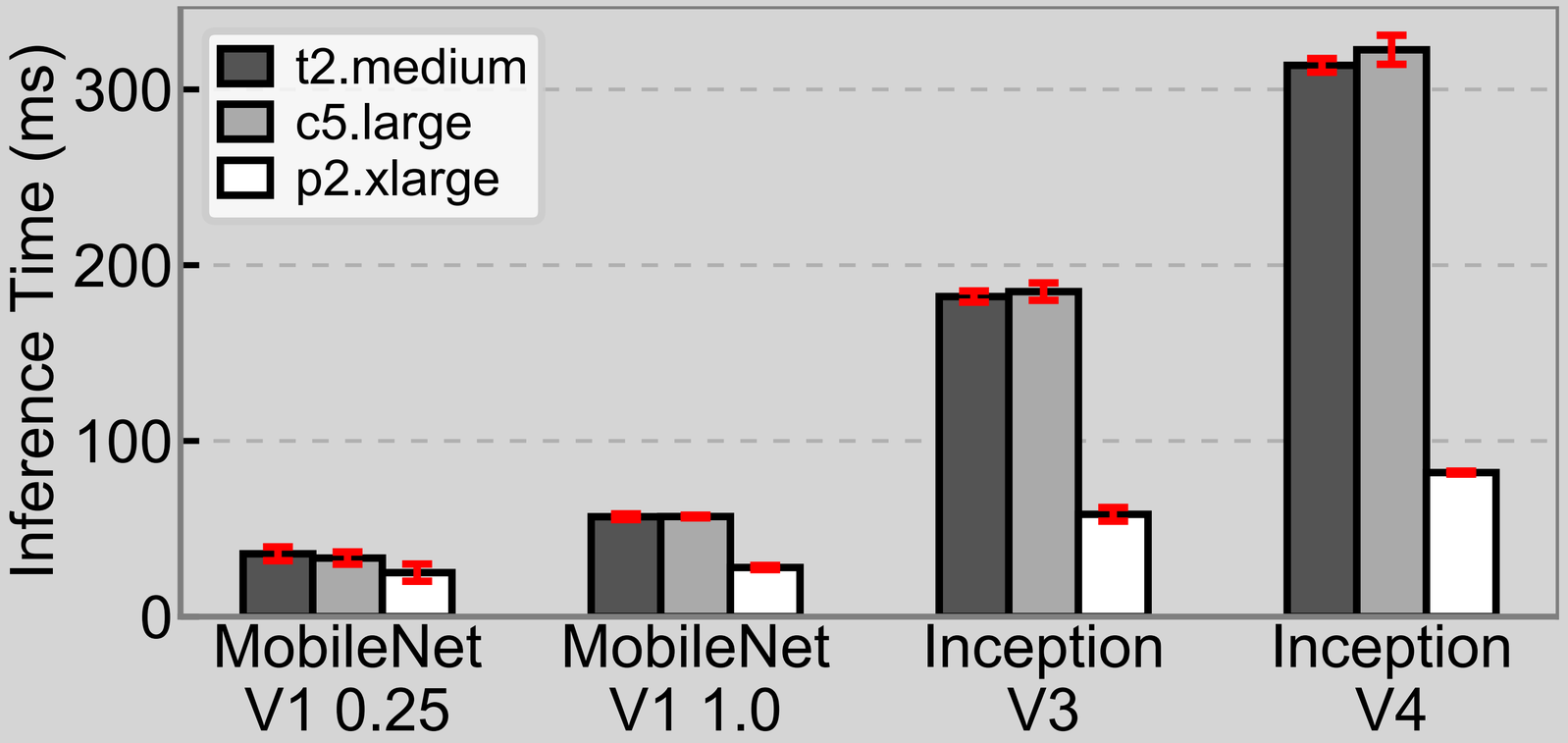
Why is it challenging?

Latency, latency



Why is it challenging?

Latency, latency, latency



How do researchers make it work?

How do researchers make it work?

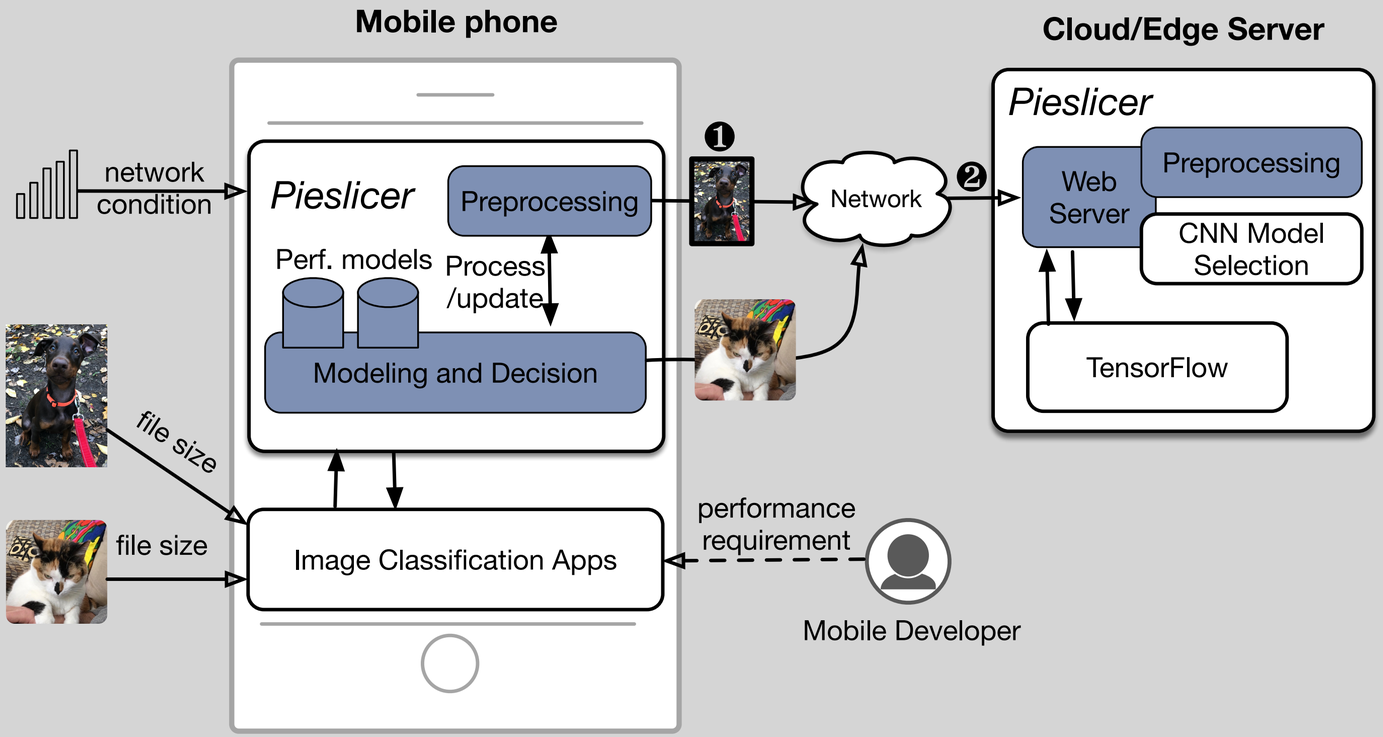
Main Approaches

- Move to cloud
- Partition models
- Shrink Models
- Redesign models

How do researchers make it work?

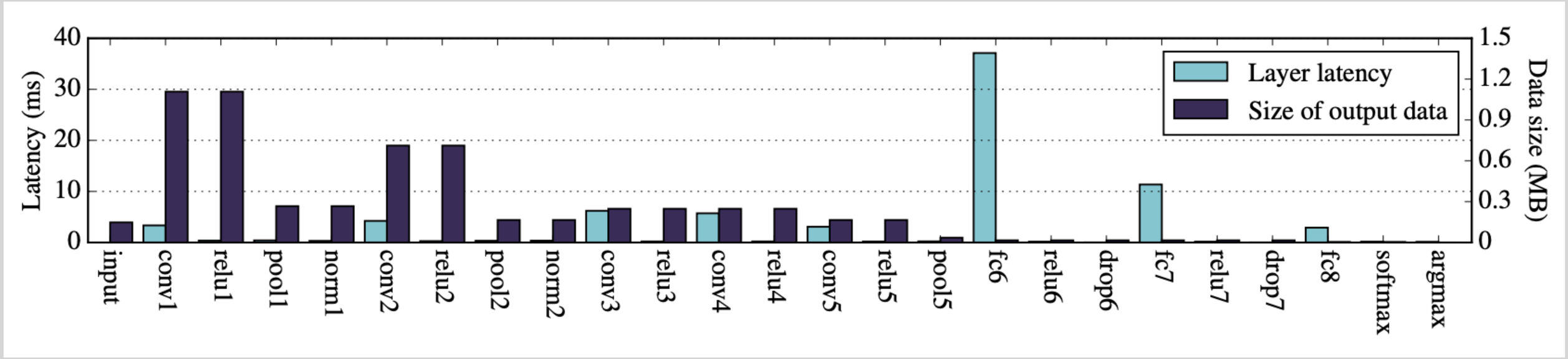
Move to the cloud

- **Pros**
  - Powerful cloud-based servers
  - Latest-and-greatest models
- **Cons**
  - Dependent on network connection
  - Large files mean high latency



How do researchers make it work?

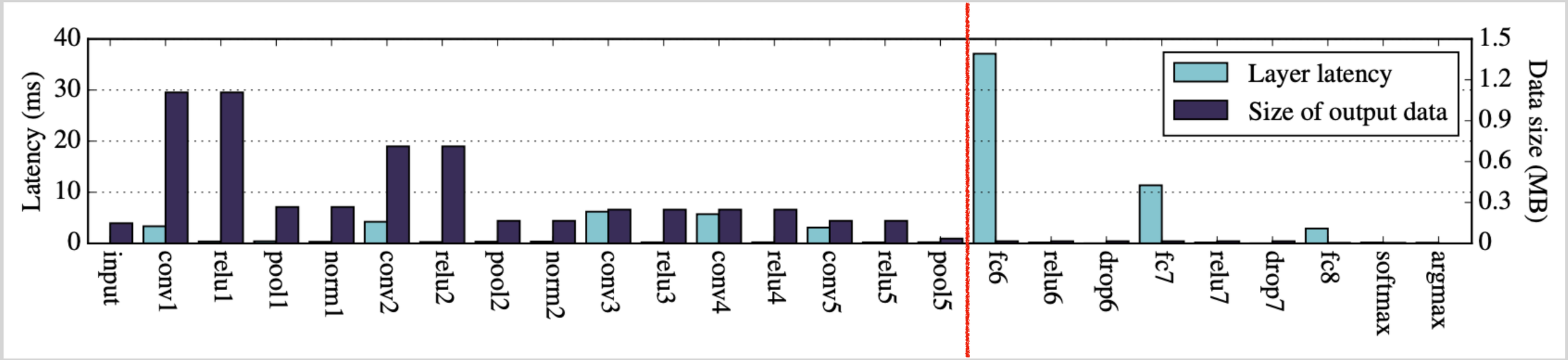
Partition Models



- **Pros**
  - Decreases data transfer
  - Improves latency and energy consumption
- **Cons**
  - Dependent on network connection for improvement
  - Only works for subset of models

How do researchers make it work?

Partition Models

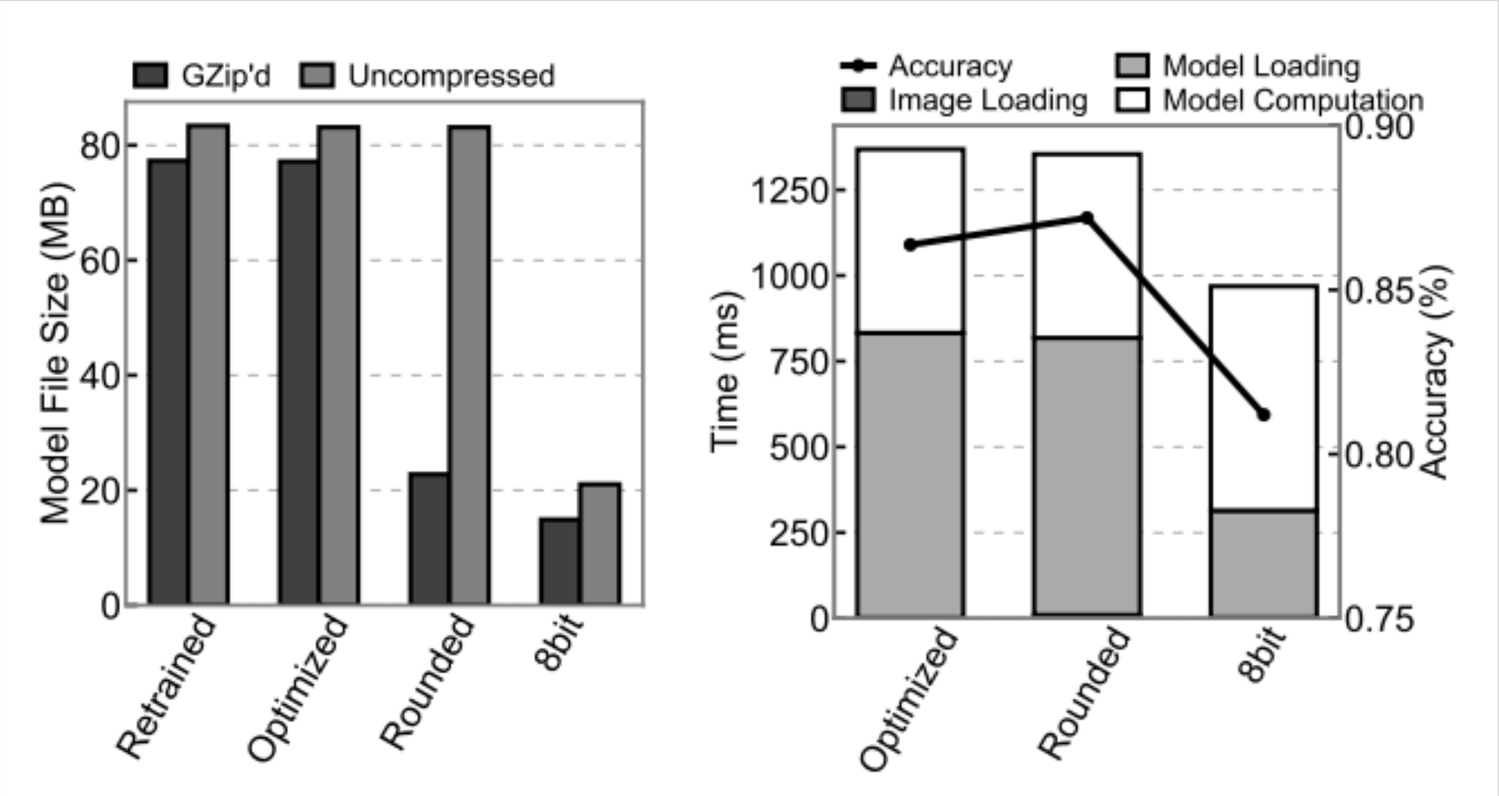


- **Pros**
  - Decreases data transfer
  - Improves latency and energy consumption
- **Cons**
  - Dependent on network connection for improvement
  - Only works for subset of models

How do researchers make it work?

Shrink Models

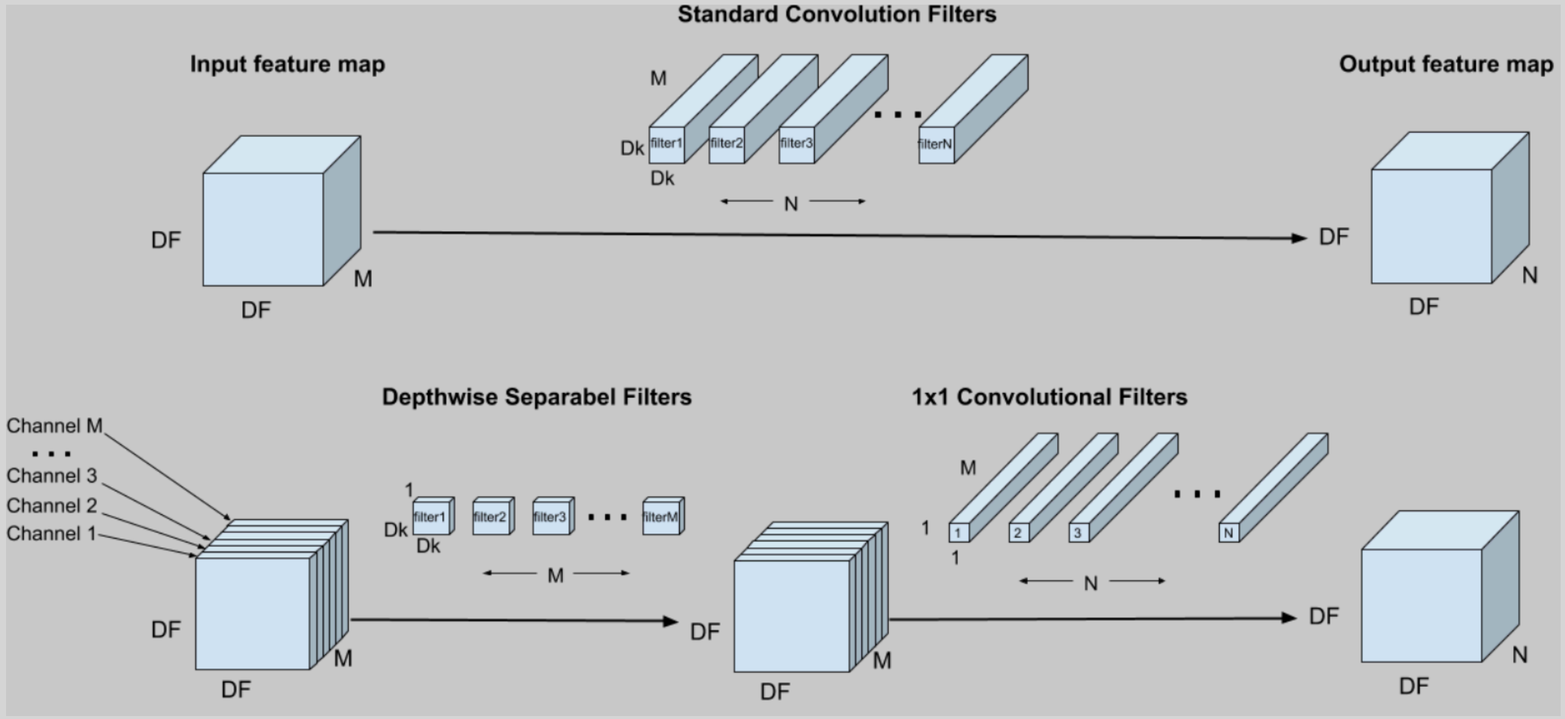
- **Pros**
  - Able to run models on-device quickly
  - Small model size on-disk and in-memory
- **Cons**
  - Decreases model accuracy
  - Can increase latency



How do researchers make it work?

Redesign Models

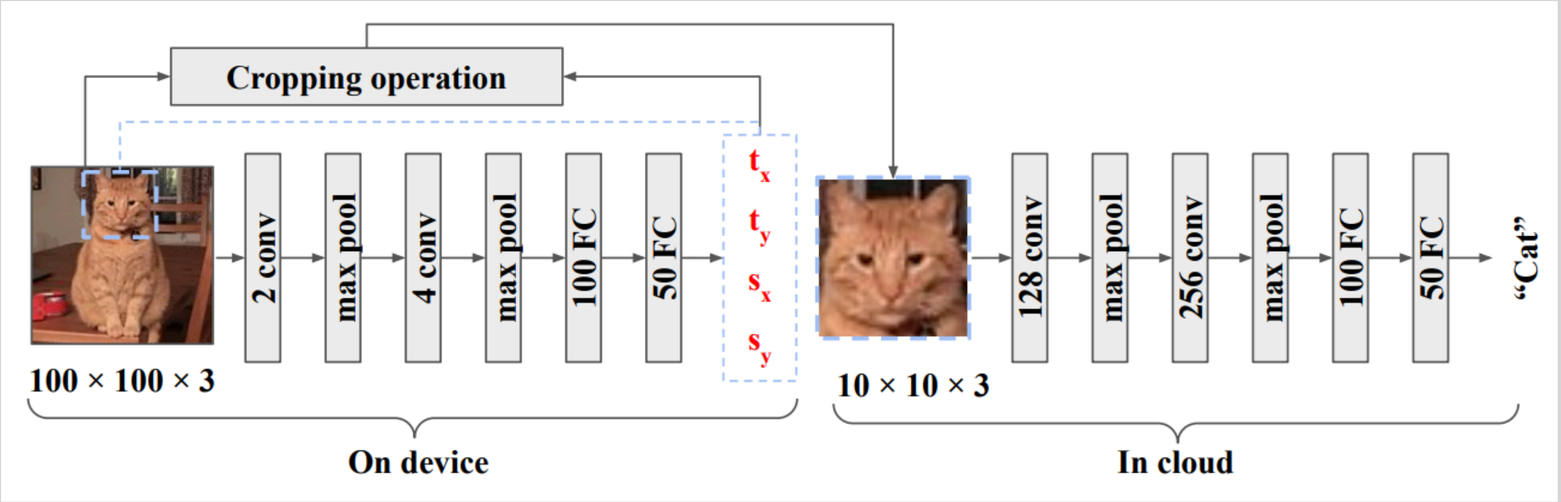
- **Pros**
  - Powerful cloud-based servers
  - Latest-and-greatest models
- **Cons**
  - Dependent on network connection
  - Large files mean high latency



How do researchers make it work?

Rethink Pipeline

- **Pros**
  - Spread computation across devices
  - Decreases network data
- **Cons**
  - Relies on network connection
  - Doesn't work for all models

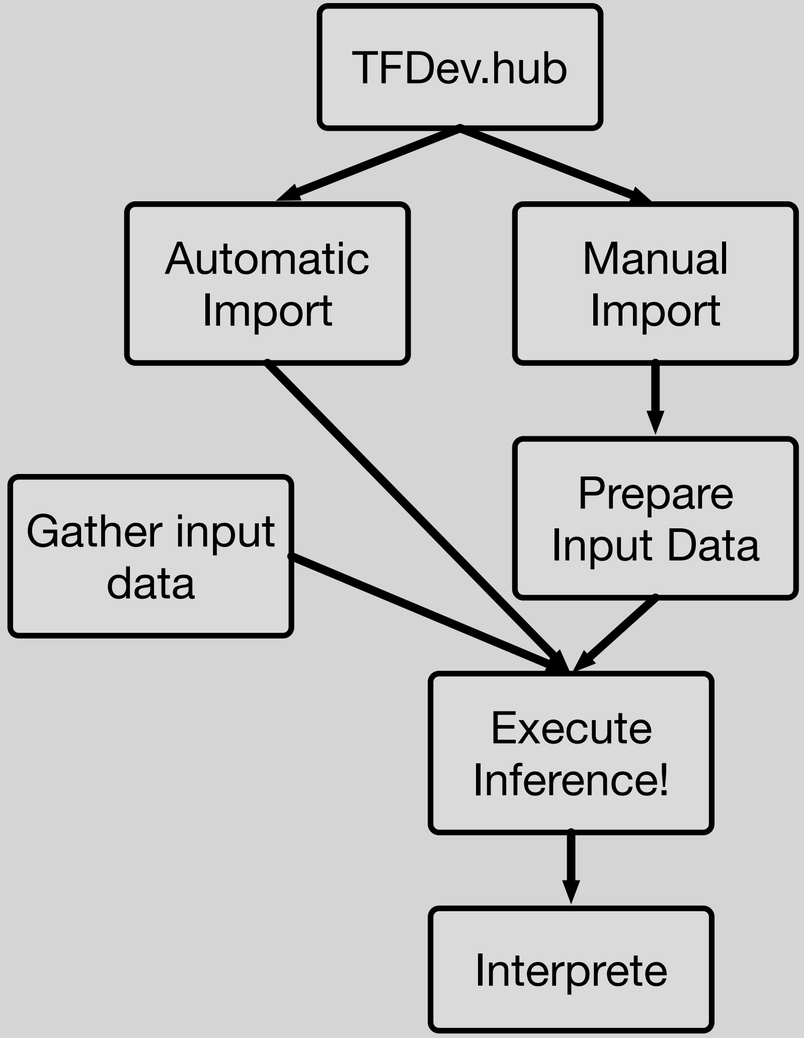


How do *we* make it work?

How do *we* make it work?

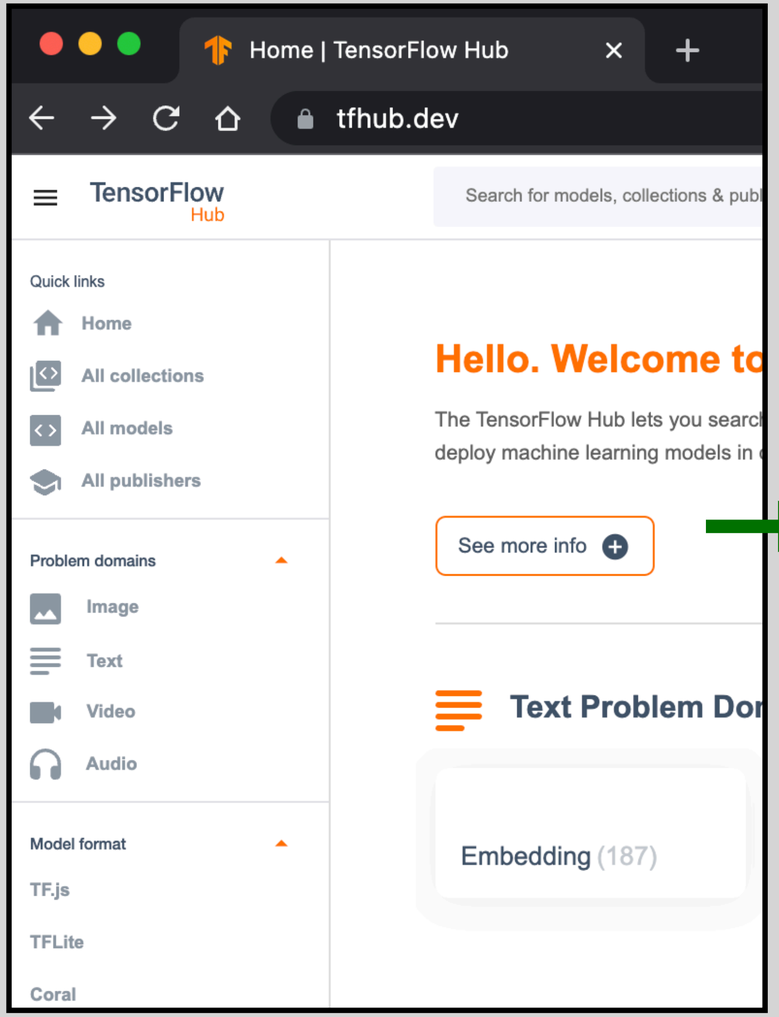
Implementation Steps Overview

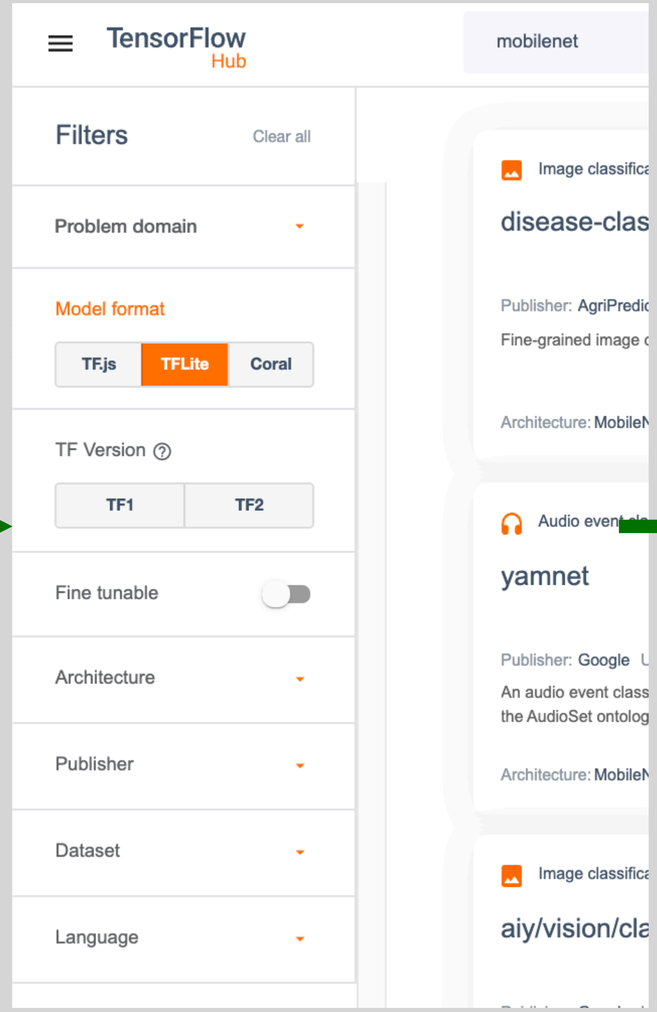
1. Find model to use
2. Prepare input data
3. Import model
4. Execute inference!
5. Interpret results



How do we make it work?

Find a model to use on tfhub.dev





How do we make it work?

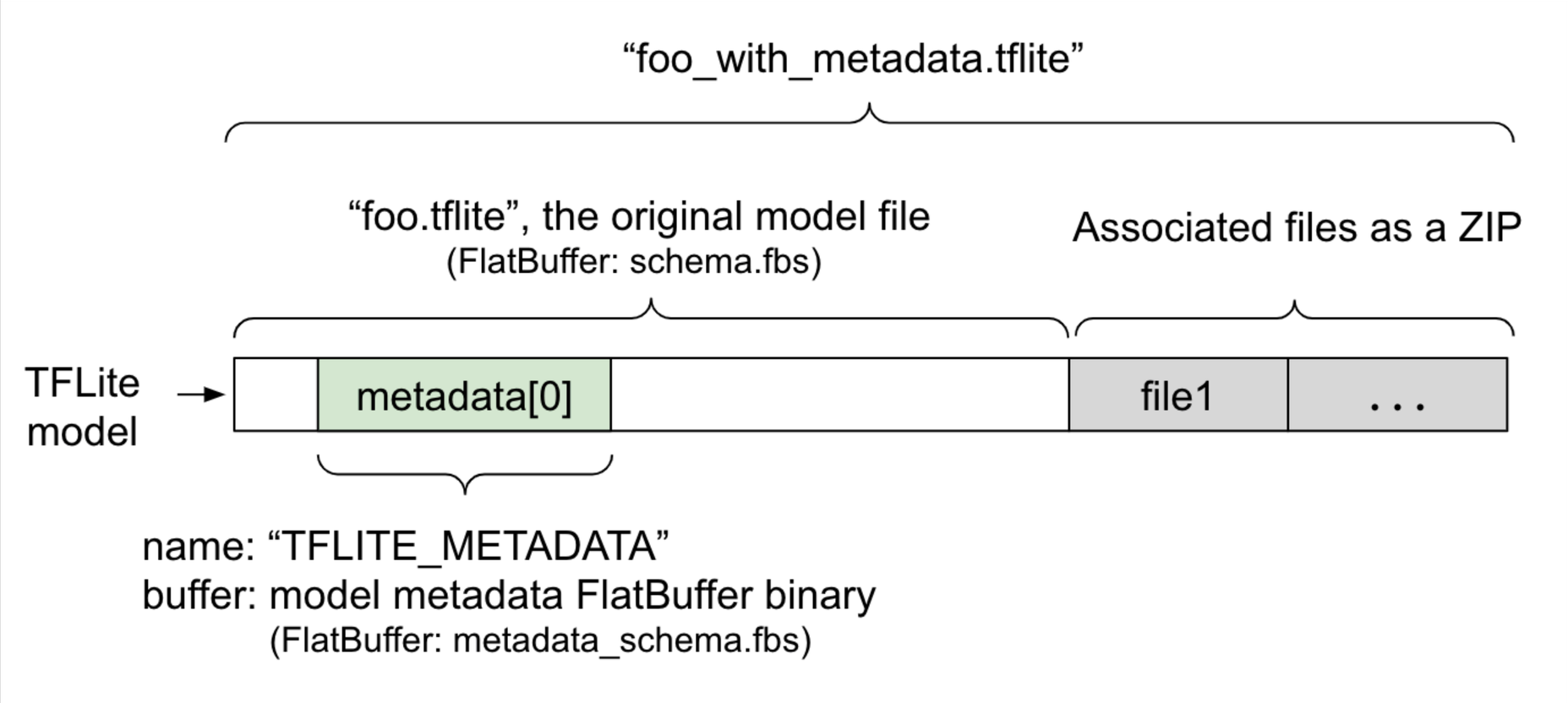
Model Details on tfhub.dev

Metadata

name	MobileNetV3 image classifier
description	Identify the most prominent object in the image from a set of 1,001 categories such as trees, animals, food, vehicles, person etc.
version	v1
author	TensorFlow
license	Apache License. Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.
min_parser_version	1.0.0
subgraph_metadata	**input_tensor_metadata**
	name: image
	description: Input image to be classified. The expected image is 224 x 224, with three channels (red, blue, and green) per pixel. Each element in the tensor is a value between min and max, where (per-channel) min is [0.0] and max is [1.0].
	stats: **max**: 1; **min**: 0
	content: **content_properties_type**: ImageProperties; **content_properties**: **color_space**: RGB
	process_units: **options_type**: NormalizationOptions; **options**: **mean**: 0

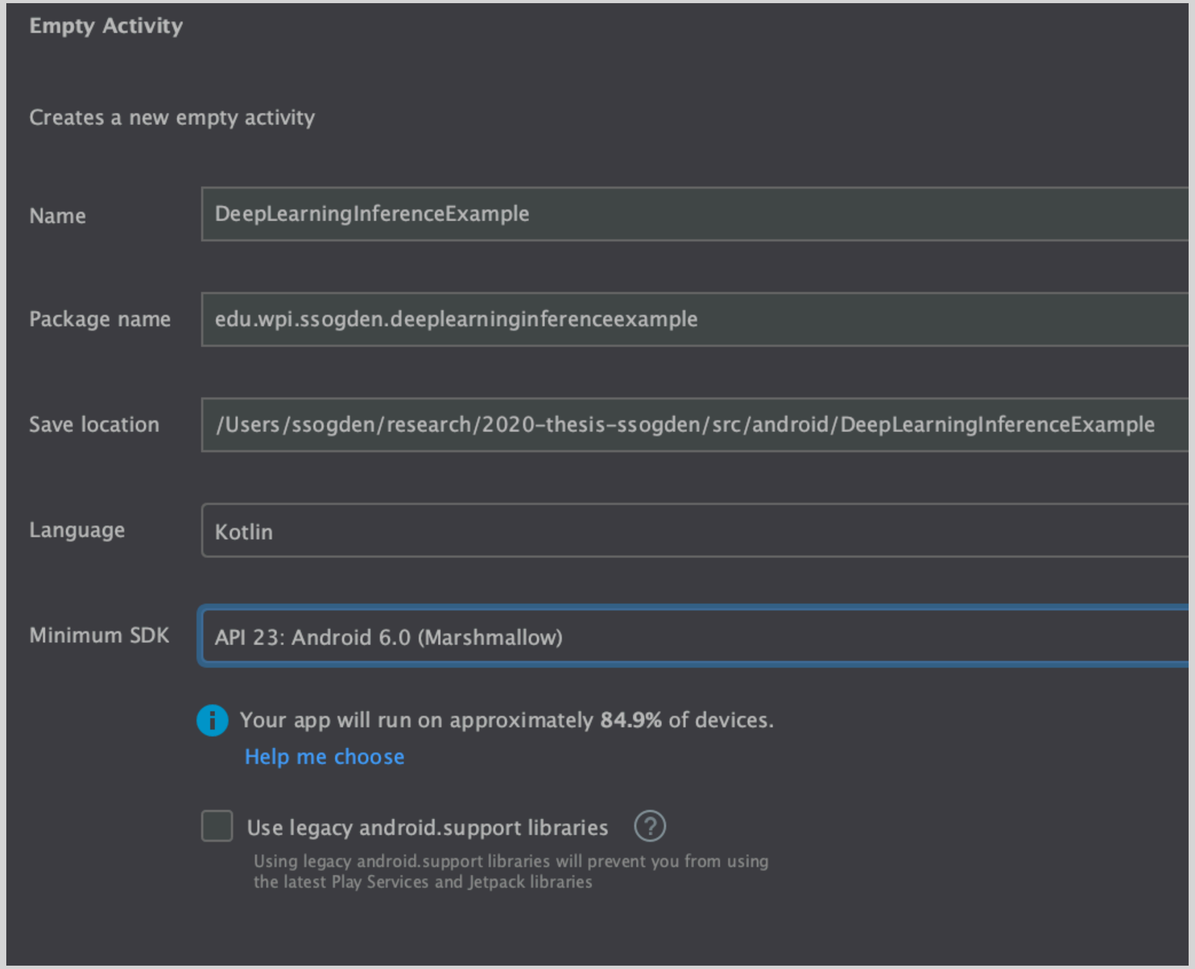
How do we make it work?

Quick aside on metadata



How do we make it work?

Setting up a project in Android Studio



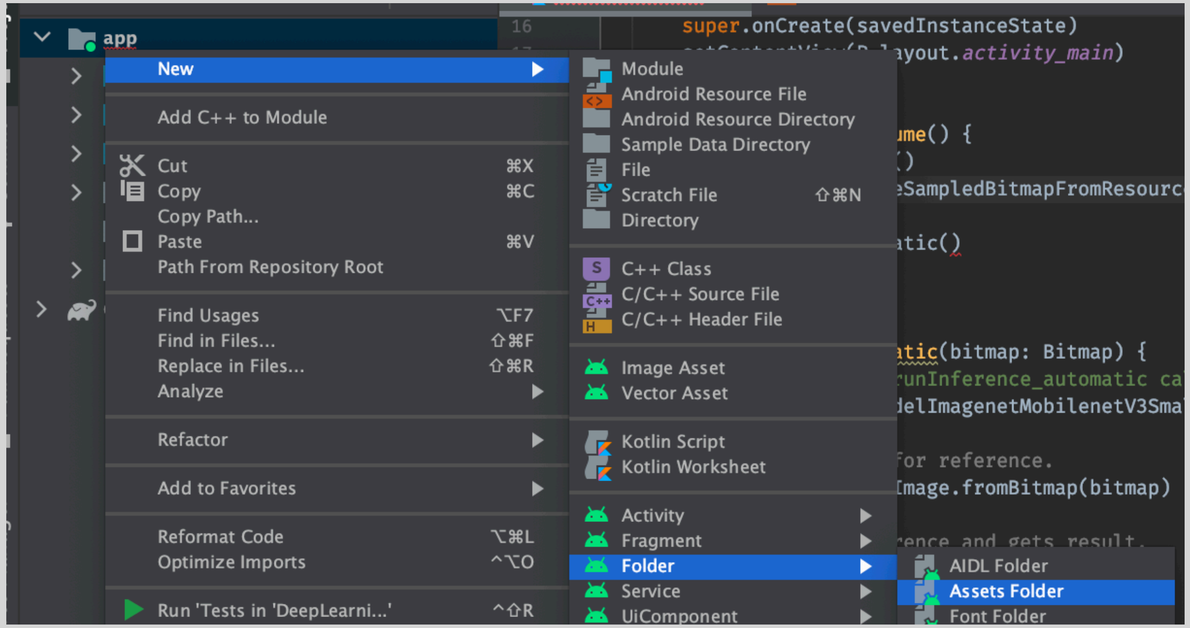
Min SDK version of TFLite

Library	minSdkVersion	Device Requirements
tensorflow-lite	19	NNAPI usage requires API 27+
tensorflow-lite-gpu	19	GLES 3.1 or OpenCL (typically only available on API 21+
tensorflow-lite-hexagon	19	-
tensorflow-lite-support	19	-
tensorflow-lite-task-vision	21	android.graphics.Color related API requires API 26+
tensorflow-lite-task-text	21	-
tensorflow-lite-task-audio	23	-
tensorflow-lite-metadata	19	-

https://www.tensorflow.org/lite/guide/android#min_sdk_version_of_tflite

How do we make it work?

Adding and loading a test image



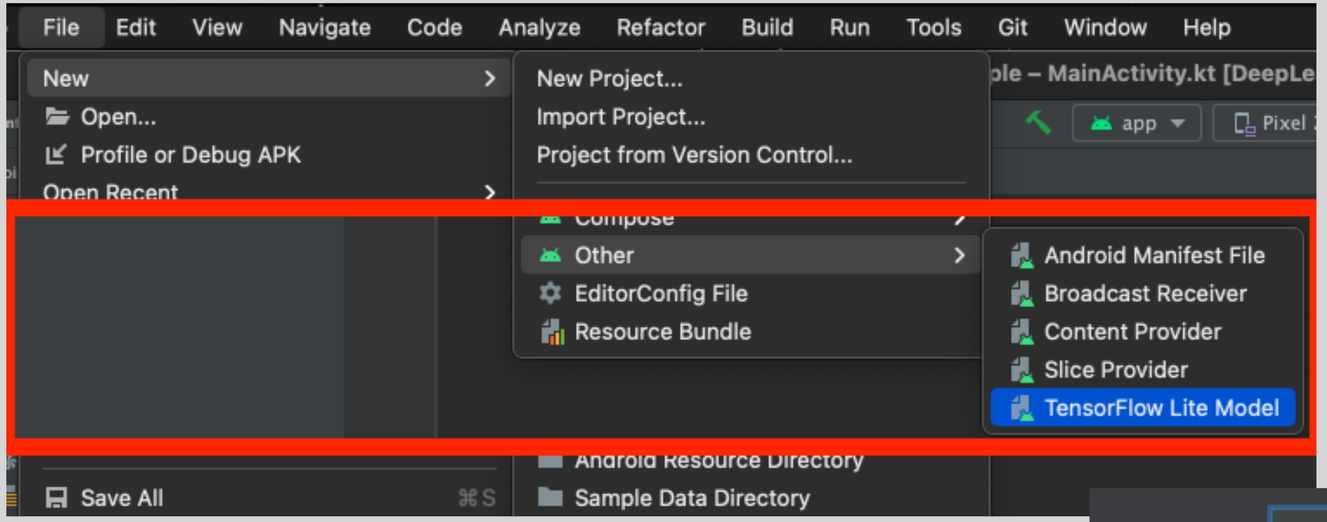

```kotlin
fun getBitmapFromAsset(context: Context, filePath: String?): Bitmap? {
    val assetManager: AssetManager = context.getAssets()
    val istr: InputStream
    var bitmap: Bitmap? = null
    try {
        istr = assetManager.open(filePath!!)
        bitmap = BitmapFactory.decodeStream(istr)
    } catch (e: IOException) {
        // handle exception
    }
    return bitmap
}
```

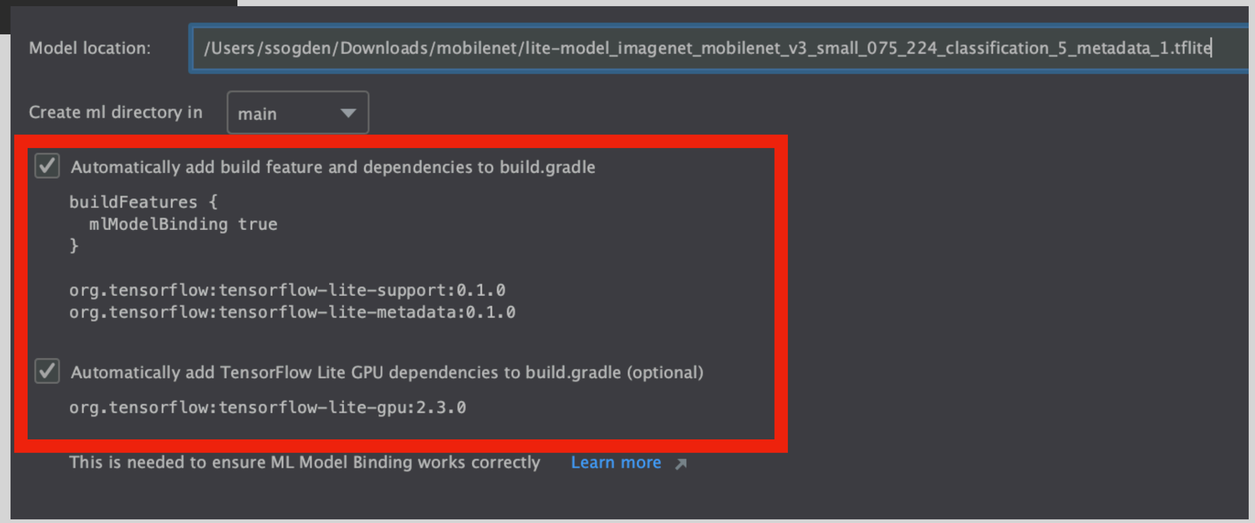
How do *we* make it work?

Automatic Setup

How do *we* make it work?

Automatic setup — Add model automatically





How do *we* make it work?

Automatic setup — Auto-filled information

Tensors

Inputs

Name	Type	Description	Shape	Min / Max
image	Image <float32>	Input image to be classified. The expected image is 224 x 224, with three channels (red, blue, and green) per pixel. Each element in the tensor is a value between min and max, where (per-channel) min is [0.0] and max is [1.0].	[1, 224, 224, 3]	[0] / [1]

Outputs

Name	Type	Description	Shape	Min / Max
logit	Feature <float32>	Logits vector of the 1001 labels respectively. Apply softmax to the logits vector to get probabilities if needed.	[1, 1001]	[] / []

Sample Code

Kotlin | Java

```kotlin
val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)

// Creates inputs for reference.
val image = TensorImage.fromBitmap(bitmap)

// Runs model inference and gets result.
val outputs = model.process(image)
val logit = outputs.logitAsCategoryList

// Releases model resources if no longer used.
model.close()
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG,  msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG,  msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG, msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG, msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG,  msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG,  msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG,  msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG,  msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG,  msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG,  msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — Execution Code

```
36      fun runInference_automatic(context: Context, bitmap: Bitmap): String {
37          Log.i(TAG,  msg: "runInference_automatic called")
38          val model = LiteModelImagenetMobilenetV3Small075224Classification5Metadata1.newInstance(context)
39
40          // Creates inputs for reference.
41          val image = TensorImage.fromBitmap(bitmap)
42
43          // Runs model inference and gets result.
44          val outputs = model.process(image)
45          val logit = outputs.logitAsCategoryList
46
47          logit.sortBy { it.score }
48          for (cat in logit) {
49              Log.d(TAG,  msg: "category: ${cat.toString()}")
50          }
51          // Releases model resources if no longer used.
52          model.close()
53          return logit.toString()
54      }
```

How do *we* make it work?

Automatic setup — How well does it work?

How do *we* do it?

Manual Setup

How do *we* make it work?

Manual setup — Steps

- Load model into ByteBuffer
- Create interpreter from ByteBuffer
- Load class dictionary
- Load into data into ByteBuffer
- Execute!
- Parse output buffer

How do *we* make it work?

Manual setup — Loading the model data

```
146     @Throws(IOException::class)
147     fun loadModelFile(assetManager: AssetManager, model_path: String?): MappedByteBuffer? {
148         assetManager.openFd(model_path!!).use { fileDescriptor ->
149             FileInputStream(fileDescriptor.fileDescriptor).use { inputStream ->
150                 val fileChannel = inputStream.channel
151                 val startOffset = fileDescriptor.startOffset
152                 val declaredLength = fileDescriptor.declaredLength
153                 return fileChannel.map(
154                     FileChannel.MapMode.READ_ONLY,
155                     startOffset,
156                     declaredLength
157                 )
158             }
159         }
160     }
```

How do *we* make it work?

Manual setup — Setting up an interpreter

```
private fun getInterpreter(context: Context, model_path: String, num_threads: Int): Interpreter {

    val options = Interpreter.Options()
    val compatList = CompatibilityList()

    if (compatList.isDelegateSupportedOnThisDevice) {
        // if the device has a supported GPU, add the GPU delegate
        val delegateOptions = compatList.bestOptionsForThisDevice
        val gpuDelegate = GpuDelegate(delegateOptions)
        options.addDelegate(gpuDelegate)
    } else {
        // if the GPU is not supported, run on given number of threads
        options.setNumThreads(num_threads)
    }
    var model: MappedByteBuffer? = null
    try {
        model = loadModelFile(context.assets, model_path)
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return Interpreter(model!!, options)
}
```

How do *we* make it work?

Manual setup — Setting up an interpreter

```
private fun getInterpreter(context: Context, model_path: String, num_threads: Int): Interpreter {

    val options = Interpreter.Options()
    val compatList = CompatibilityList()

    if (compatList.isDelegateSupportedOnThisDevice) {
        // if the device has a supported GPU, add the GPU delegate
        val delegateOptions = compatList.bestOptionsForThisDevice
        val gpuDelegate = GpuDelegate(delegateOptions)
        options.addDelegate(gpuDelegate)
    } else {
        // if the GPU is not supported, run on given number of threads
        options.setNumThreads(num_threads)
    }
    var model: MappedByteBuffer? = null
    try {
        model = loadModelFile(context.assets, model_path)
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return Interpreter(model!!, options)
}
```

How do *we* make it work?

Manual setup — Setting up an interpreter

```
private fun getInterpreter(context: Context, model_path: String, num_threads: Int): Interpreter {

    val options = Interpreter.Options()
    val compatList = CompatibilityList()

    if (compatList.isDelegateSupportedOnThisDevice) {
        // if the device has a supported GPU, add the GPU delegate
        val delegateOptions = compatList.bestOptionsForThisDevice
        val gpuDelegate = GpuDelegate(delegateOptions)
        options.addDelegate(gpuDelegate)
    } else {
        // if the GPU is not supported, run on given number of threads
        options.setNumThreads(num_threads)
    }
    var model: MappedByteBuffer? = null
    try {
        model = loadModelFile(context.assets, model_path)
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return Interpreter(model!!, options)
}
```

How do *we* make it work?

Manual setup — Setting up an interpreter

```
private fun getInterpreter(context: Context, model_path: String, num_threads: Int): Interpreter {

    val options = Interpreter.Options()
    val compatList = CompatibilityList()

    if (compatList.isDelegateSupportedOnThisDevice) {
        // if the device has a supported GPU, add the GPU delegate
        val delegateOptions = compatList.bestOptionsForThisDevice
        val gpuDelegate = GpuDelegate(delegateOptions)
        options.addDelegate(gpuDelegate)
    } else {
        // if the GPU is not supported, run on given number of threads
        options.setNumThreads(num_threads)
    }
    var model: MappedByteBuffer? = null
    try {
        model = loadModelFile(context.assets, model_path)
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return Interpreter(model!!, options)
}
```

How do *we* make it work?

Manual setup — Loading Class Dictionary

```
$ nl labels.txt  | head -20
     1  background
     2  tench
     3  goldfish
     4  great white shark
     5  tiger shark
     6  hammerhead
     7  electric ray
     8  stingray
     9  cock
    10  hen
    11  ostrich
    12  brambling
    13  goldfinch
    14  house finch
    15  junco
    16  indigo bunting
    17  robin
    18  bulbul
    19  jay
    20  magpie
```

```
182     fun loadDictionary(
183         assetManager: AssetManager,
184         labels_path: String
185     ): MutableMap<Int, String> {
186         var dic: MutableMap<Int, String> = HashMap()
187
188         var reader = BufferedReader(
189             InputStreamReader(getAssets().open(labels_path),
190                 charsetName: "UTF-8"))
191
192         var curr_idx = 0
193         for (line in reader.readLines()) {
194             dic.put(curr_idx++, line)
195         }
196         return dic
197     }
```

How do *we* make it work?

Manual setup — Load input data

```
162     private fun convertBitmapToByteBuffer(bitmap: Bitmap): ByteBuffer {
163         val width = 224 // Determined by model
164         val height = 224 // Determined by model
165         val num_channels = 3 // Determined by bitmap
166         val byteBuffer = ByteBuffer.allocateDirect( capacity: width * height * num_channels * 4) // WIDTH * HEIGHT * PIXEL_SIZE
167         byteBuffer.order(ByteOrder.nativeOrder())
168         val intValues = IntArray( size: width * height)
169         bitmap.getPixels(intValues,  offset: 0, bitmap.width,  x: 0,  y: 0, bitmap.width, bitmap.height)
170         var pixel = 0
171         for (i in 0 until width) {
172             for (j in 0 until height) {
173                 val `val` = intValues[pixel++]
174                 byteBuffer.putFloat( ((`val` shr 16 and 0xFF) / 255.0f) ) // Channel 1
175                 byteBuffer.putFloat( ((`val` shr 8 and 0xFF) / 255.0f) ) // Channel 2
176                 byteBuffer.putFloat( ((`val` and 0xFF) / 255.0f) ) // Channel 3
177             }
178         }
179         return byteBuffer
180     }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in,  dstWidth: 224,  dstHeight: 224,  filter: false)
82        val dic = loadDictionary(context.assets,  labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in, dstWidth: 224, dstHeight: 224, filter: false)
82        val dic = loadDictionary(context.assets, labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in, dstWidth: 224, dstHeight: 224, filter: false)
82        val dic = loadDictionary(context.assets, labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in, dstWidth: 224, dstHeight: 224, filter: false)
82        val dic = loadDictionary(context.assets, labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in,  dstWidth: 224,  dstHeight: 224,  filter: false)
82        val dic = loadDictionary(context.assets,  labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — Execute it!

```
79    fun runInference_manual(context: Context, model_path: String, bitmap_in: Bitmap, num_threads: Int): String {
80
81        val bitmap = Bitmap.createScaledBitmap(bitmap_in, dstWidth: 224, dstHeight: 224, filter: false)
82        val dic = loadDictionary(context.assets, labels_path: "labels.txt")
83
84        var interpreter = getInterpreter(context, model_path, num_threads)
85        interpreter.allocateTensors()
86
87        var input_buffer = convertBitmapToByteBuffer(bitmap)
88
89        var output_buffer_size = 1001 // num_categories
90        //var output_buffer_size = interpreter.getOutputTensor(0).numBytes()
91        var output_buffer = FloatBuffer.allocate(output_buffer_size)
92
93
94        val inputs = arrayOf(input_buffer)
95        val outputs = mapOf(0 to output_buffer)
96        interpreter.runForMultipleInputsOutputs(inputs, outputs)
97
98        var output_arr = output_buffer.array()
99
100       interpreter.close()
101       return dic[output_arr.indices.maxByOrNull { output_buffer[it] }].toString();
102
103   }
```

How do *we* make it work?

Manual setup — How well does it work?

How do *we* make it work?

So why use manual?

How do *we* make it work?

So why use manual?

Pros of manual

- More flexibility!
- Faster!
  - 23.747ms vs 28.442ms
  - 26.938ms on GPU without reload vs 198.627ms with reload

Cons of manual

- More complicated set up of model file
- More manual fiddling of bits (think bitmap to ByteBuffer)
- Manual output

Wrap-up

- Uses of deep learning
- Challenges of deep learning inference
- Ways researchers have addressed these challenges
- Implementing deep learning in an application
  - Comparison of approaches and their benefits

Resources

- Example Application on Github
  - https://github.com/cake-lab/MobileDeepInferenceExample
- Models to use on-device (and sometimes in-cloud)
  - https://tfhub.dev/
- TFLite homepage
  - https://www.tensorflow.org/lite