Machine Learning Developer Day

January 24, 2019
12:00-5:00pm

C L Kim

Verizon Media (FKA Oath Inc./Aol/Yahoo)

USING ML IN AN ANDROID DEMO APP - THE GOOD, THE BAD, AND THE UGLY

OVERVIEW

- Code for the demo app on github github.com/clkim/NumberDetector
    - Demo app simply tries to classify hand-drawn digits
- This talk highlights some of the Good, the Bad, and the Ugly
    - Journey using a trained ML model to build an Android demo app
    - Some code (Kotlin), and some details of ML model needed to develop app
        - Time constraint: only covering code related to working with the ML model
- We do not cover in any depth:
    - Building/Training a ML model (Google TensorFlow Lite)
    - APIs: Android; Firebase (Google's mobile dev platform: iOS, Android, Web)

ACKNOWLEDGEMENTS AND CREDITS

- Mark Allison: *ML for Android Developers* – Part 1, 2, 3
  - The inspiration and material for this talk, as well as the source code for the demo app, are largely from the above blog series blog.stylingandroid.com/ml-for-android-developers-part-1-2/
- Tianxing Li: github repo *MNIST with TensorFlow Lite on Android*
  - The TensorFlow model trained on the MNIST dataset, and converted to TensorFlow Lite github.com/nex3z/tflite-mnist-android
    - Uses *tf.nn* instead of *tf.keras.layers* module used by TensorFlow "official" MNIST model; seems equivalent

SOME KOTLIN (1 / 3)

- Kotlin is JVM language that Google seems to promote for Android development developer.android.com/kotlin/
- “Pascal syntax” so the type comes after name and colon

```
fun classify(
        bitmap: Bitmap,
        success: (Int, Float, Long) -> Unit
)
```

SOME KOTLIN (2 / 3)

- Extension Functions with Receiver Type (inspired by C#)

```
private val imagePixels = IntArray(imageSize)
...

private fun Bitmap.toVector(): Array<Array<Array<FloatArray>>> {
    getPixels(imagePixels, 0, width, 0, 0, width, height)
    return Array(1) {
        Array(imageHeight) { y ->
            Array(imageWidth) { x ->
                floatArrayOf(imagePixels[x + (y * imageWidth)]
                        .convertToGreyScale()
                )
            }
        }
    }
}
```

SOME KOTLIN (3 / 3)

- Safe Calls (null safety) with safe call operator ?.

```
interpreter.run(inputs, options)
        .addOnSuccessListener { outputs ->
            outputs.map().entries.maxBy { it.value }
                ?.also { maxEntry ->
                    success(maxEntry.key, maxEntry.value,
                            System.currentTimeMillis() - start)
                }
        }
```

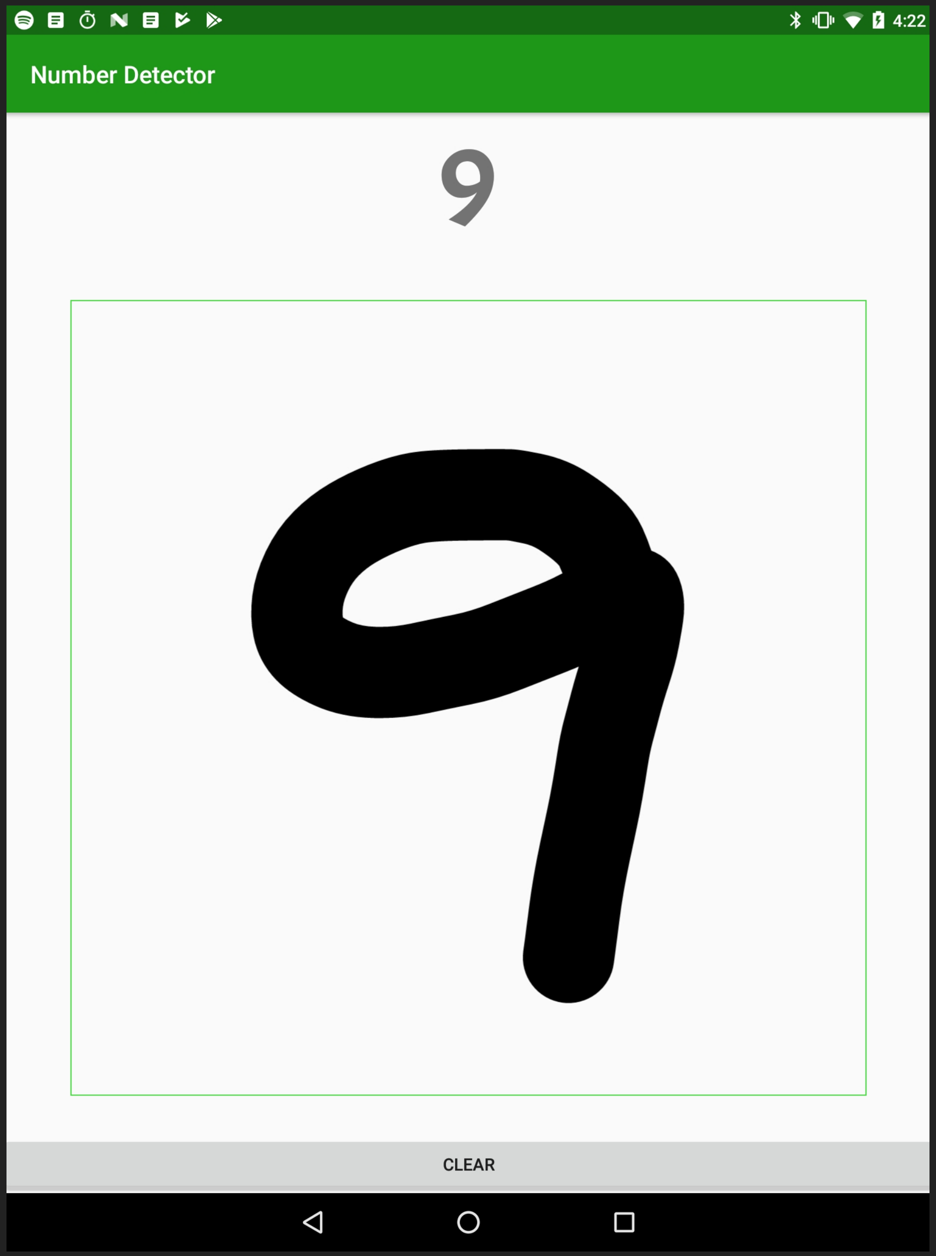
- maxEntry?. also({<*lambda*>})
  - also({<*lambda*>}) is not called if maxEntry == null, no NPE;
  - also({<*lambda*>}) is called otherwise

SOME ML FRAMEWORKS/LIBRARIES

- Mobile focused:
  - Google TensorFlow Lite, Apple Core ML, Caffe2(Facebook), Xmartlabs Bender, Quantized-CNN
- Server focused:
  - TensorFlow, Amazon AML, Microsoft Cognitive Toolkit, Apache MXNet, Apache Spark MLlib

- Source: http://hopinfirst.com/top-10-machine-learning-frameworks-mobile-apps/

SIMPLE ANDROID DEMO APP



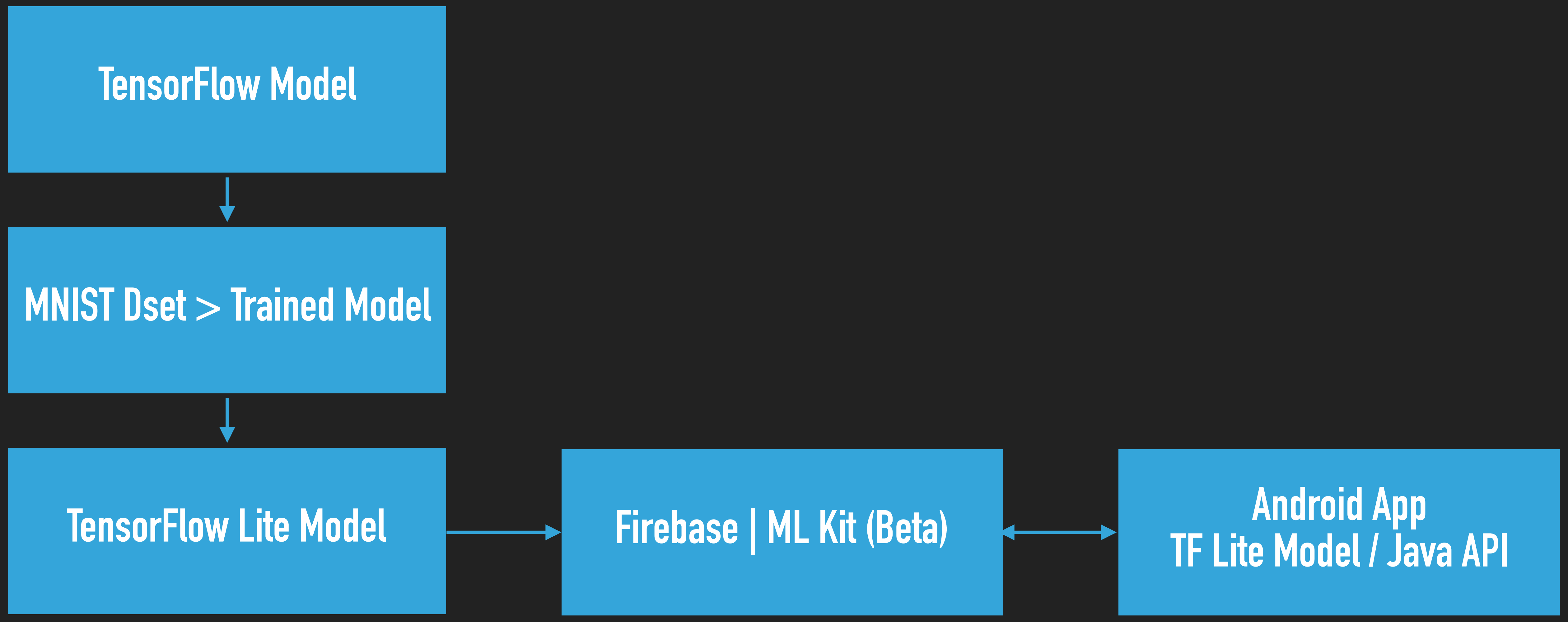
Hand-draw a digit, multiple motions ok

Acknowledgement:

Mark Allison's blog posts and sample code is the inspiration and source for this talk

https://blog.stylingandroid.com/ml-for-android-developers-part-1-2/

OVERVIEW - ANDROID DEMO APP USING MACHINE LEARNING



TensorFlow Model

MNIST Dset > Trained Model

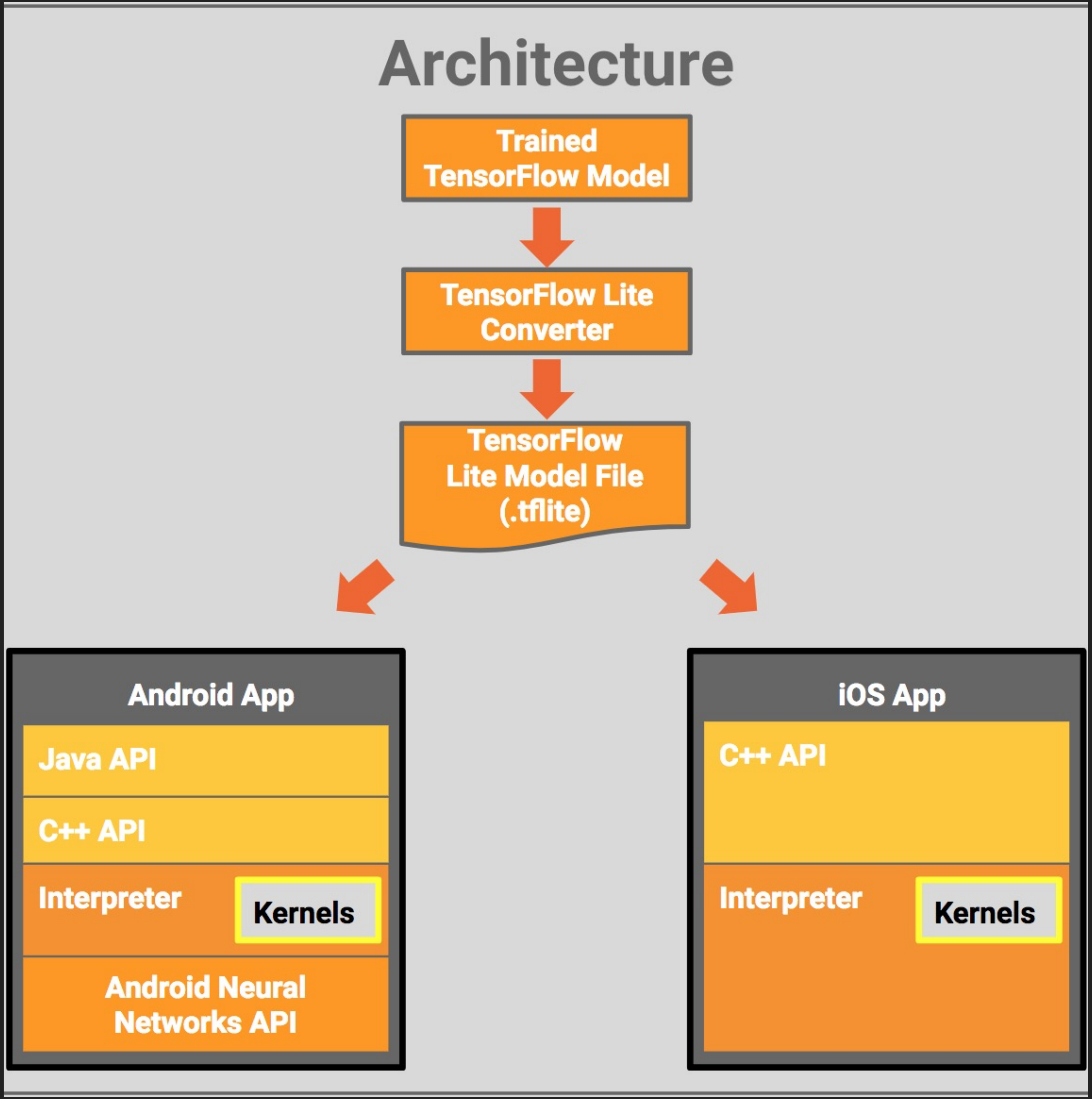
TensorFlow Lite Model

Firebase | ML Kit (Beta)

Android App
TF Lite Model / Java API

TENSORFLOW LITE

- TensorFlow for mobile/ embedded devices
- Low latency, small binary
- Android iOS RaspberryPi
- On-device interpreter
- Supports hw acceleration
- Ref: TensorFlow Lite > GUIDE tab (TFLG) www.tensorflow.org/lite/overview



USE TF LITE MODEL DIRECTLY, OR HOSTED BY ML KIT (BETA)

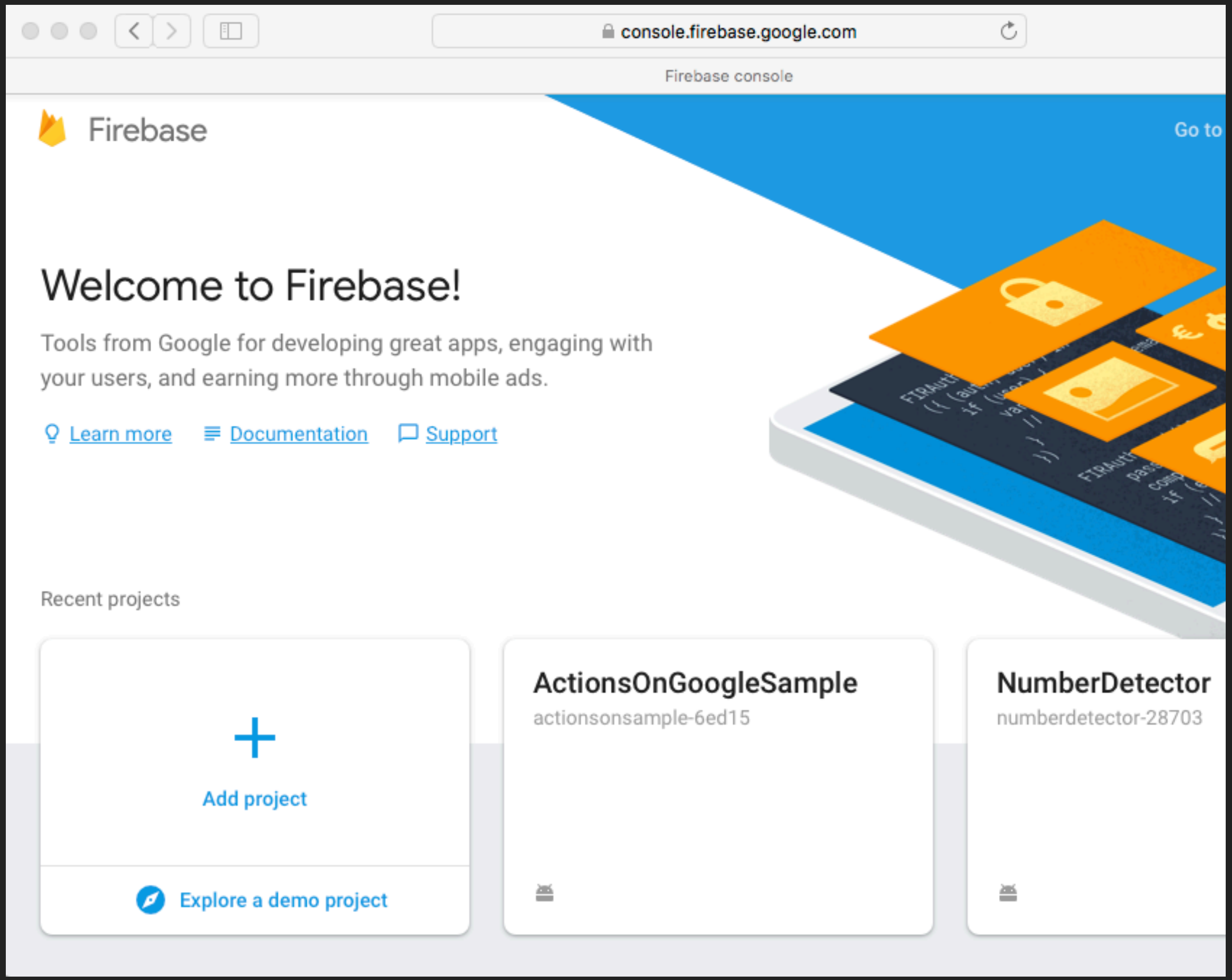
- TFLG > Android demo app: camera demo app; classifies images continuously captured by camera
  - The build process downloads the trained model, eg. quantized MobileNet TensorFlow Lite, and bundles it into the app binary
- We use ML Kit (beta release) for Firebase to host trained model
  - Decouple the model from the app
  - The Good: An upgraded model can be released at any time, without an app update to Google Play (or Apple App Store)

GETTING A TF LITE MODEL

- TFLG > Developer guide: good overview of
  - Training a pre-trained or custom model
  - Converting to TensorFlow Lite model
- Our Demo App takes a short-cut: it does use the MNIST dataset of hand-drawn numeric digits; but instead of starting from scratch, we found a trained model already converted to TensorFlow Lite, e.g. *mnist.tflite*
  (by Tianxing Li, github.com/nex3z/tflite-mnist-android)

ML KIT

- First, create a Firebase account: login at console.firebase.google.com
- > Add project, e.g. NumberDetector
- > Download google-services.json > follow paged instructions to update the app project in Android Studio



Welcome to Firebase!

Tools from Google for developing great apps, engaging with your users, and earning more through mobile ads.

Recent projects

ActionsOnGoogleSample
actionsonsample-6ed15

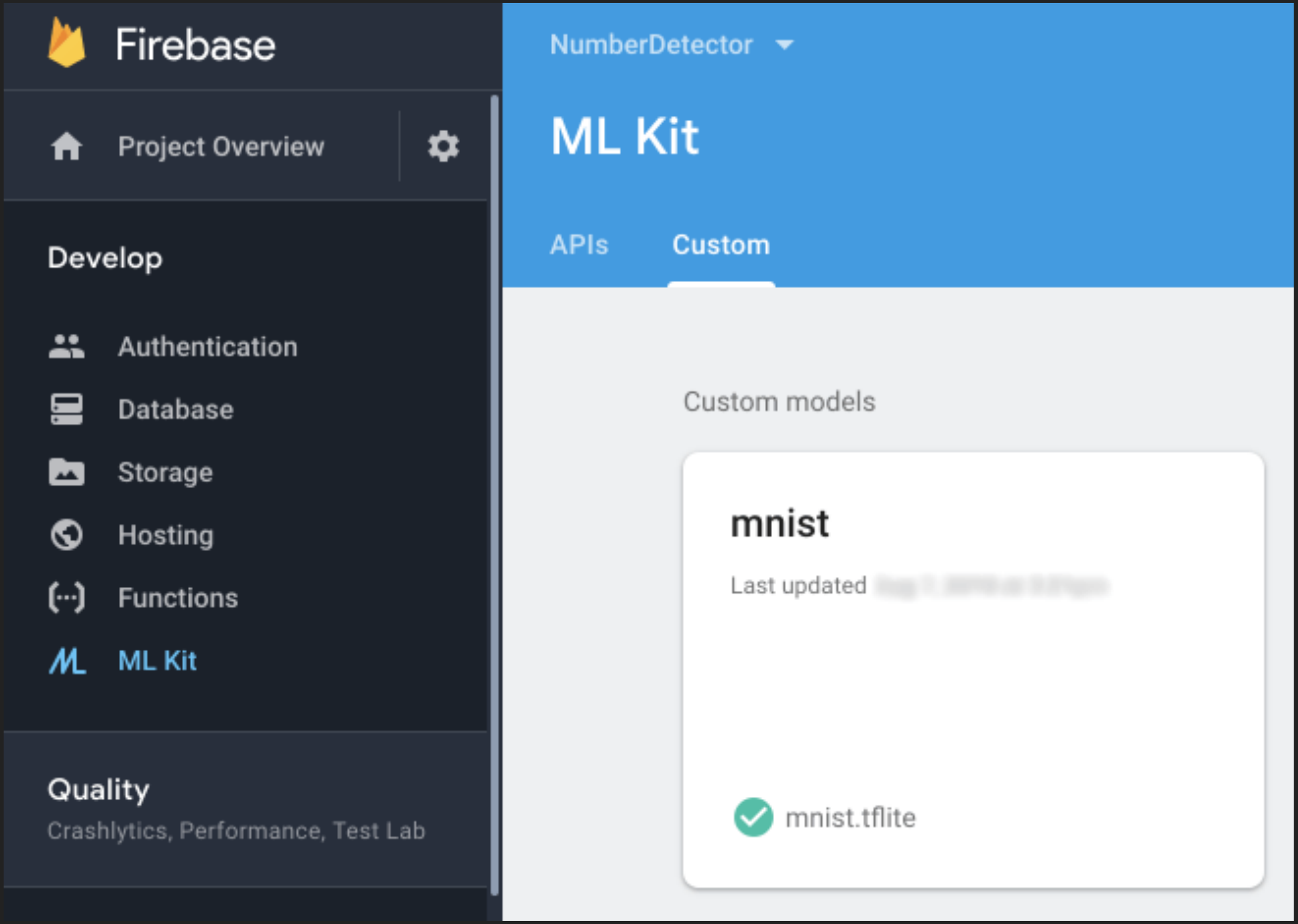
NumberDetector
numberdetector-28703

ML KIT (2)

- Last step: connect the app to Firebase by running the app from IDE > Run | Run app; but Catch-22 is: what if app is not ready/debugged?
    - A hack, try: use Firebase Assistant to pick any (benign) Firebase feature in order to connect the app, so from IDE > Tool | Firebase > e.g. Test Lab > Run Firebase Test Lab…

ML KIT (3)

- Upload the *mnist.tflite* file to Firebase
- Firebase console > ML Kit > Custom tab



Firebase console

MAGIC SAUCE - THE INPUT AND OUTPUT FORMATS

- A TF Lite model expects as input, and produces as output
  - One or more multidimensional arrays
  - Values of type: *byte*, *int*, *long*, or *float*
- The Ugly: How to know what are their “shape” and type?
  - 1. Ask the Machine Learning team which trained the model and provided the *.tflite* file
  - 2. Find out from the *.tflite* file

GRAPH VISUALIZATION TOOL

- There is tool to visualize TensorFlow Lite models…
  - if you can find/build it :)

1. Install Bazel build tool, see: docs.bazel.build/versions/master/install.html

2. Clone the TensorFlow repo, run: git clone *https://github.com/tensorflow/tensorflow.git*

3. $ cd tensorflow

4. Stay in subdir; first, confirm file's path has not changed: *tensorflow/lite/tools/visualize.py*

5. then run: $ bazel build tensorflow/lite/tools/visualize (~1 hour on my 2GHz MacBookPro)

6. then run: $ bazel-bin/tensorflow/lite/tools/visualize <path-to>/mnist.tflite <path-to>/mnist.html (creates the *mnist.html* file at specified <path-to> location)

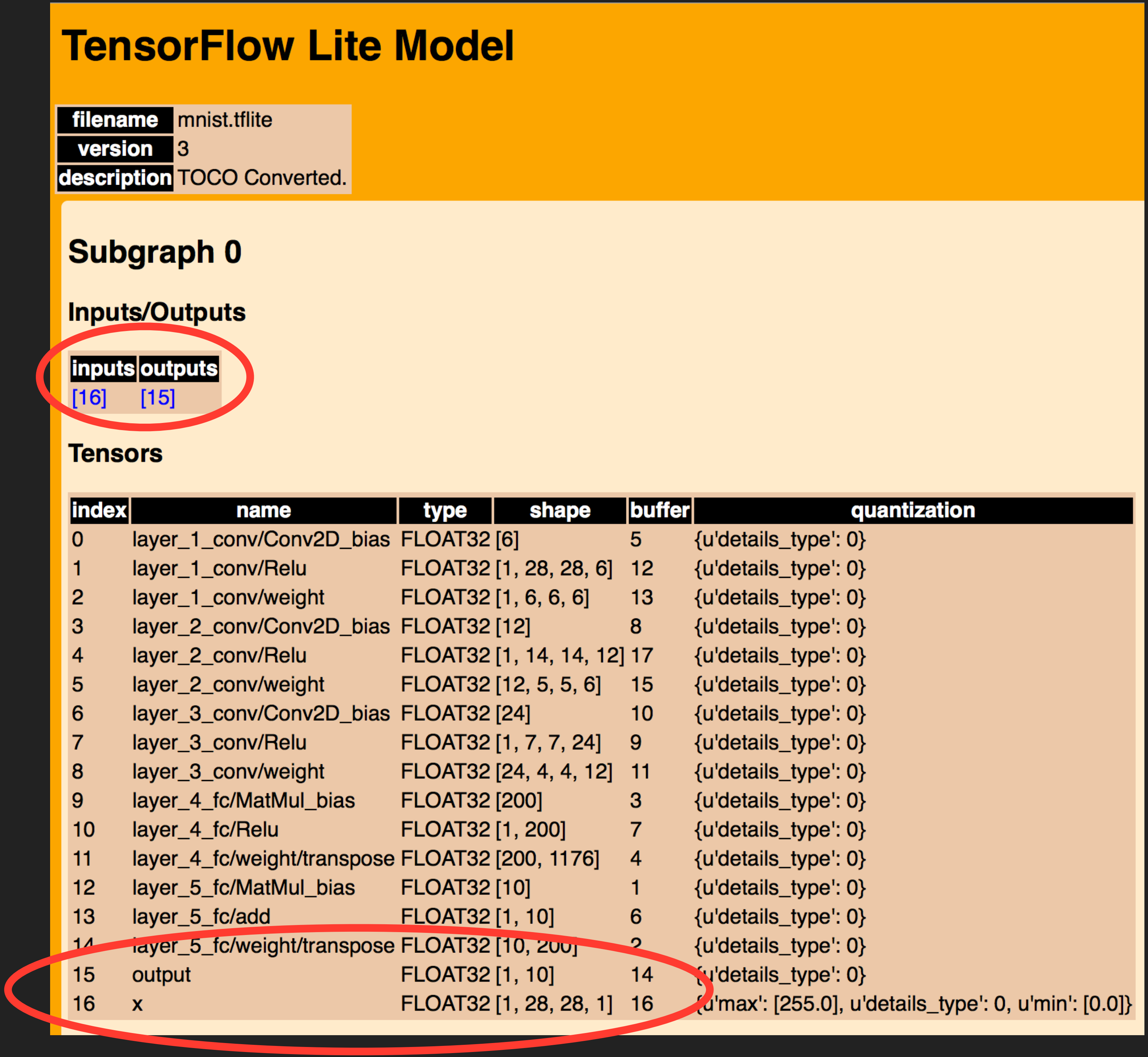
7. Note: didn't try but maybe can combine steps 5/6, see www.tensorflow.org/lite/devguide

INPUT AND OUTPUT FORMATS IN A TENSOR FLOW LITE MODEL

- Open the generated mnist.html file

- inputs: index 16
  shape [1, 28, 28, 1]
  type FLOAT32
  name "x"

- outputs: index 15
  shape [1, 10]
  type FLOAT32
  name "output"



TensorFlow Lite Model

filename	mnist.tflite
version	3
description	TOCO Converted.

Subgraph 0

Inputs/Outputs

inputs	outputs
[16]	[15]

Tensors

index	name	type	shape	buffer	quantization
0	layer_1_conv/Conv2D_bias	FLOAT32	[6]	5	{u'details_type': 0}
1	layer_1_conv/Relu	FLOAT32	[1, 28, 28, 6]	12	{u'details_type': 0}
2	layer_1_conv/weight	FLOAT32	[1, 6, 6, 6]	13	{u'details_type': 0}
3	layer_2_conv/Conv2D_bias	FLOAT32	[12]	8	{u'details_type': 0}
4	layer_2_conv/Relu	FLOAT32	[1, 14, 14, 12]	17	{u'details_type': 0}
5	layer_2_conv/weight	FLOAT32	[12, 5, 5, 6]	15	{u'details_type': 0}
6	layer_3_conv/Conv2D_bias	FLOAT32	[24]	10	{u'details_type': 0}
7	layer_3_conv/Relu	FLOAT32	[1, 7, 7, 24]	9	{u'details_type': 0}
8	layer_3_conv/weight	FLOAT32	[24, 4, 4, 12]	11	{u'details_type': 0}
9	layer_4_fc/MatMul_bias	FLOAT32	[200]	3	{u'details_type': 0}
10	layer_4_fc/Relu	FLOAT32	[1, 200]	7	{u'details_type': 0}
11	layer_4_fc/weight/transpose	FLOAT32	[200, 1176]	4	{u'details_type': 0}
12	layer_5_fc/MatMul_bias	FLOAT32	[10]	1	{u'details_type': 0}
13	layer_5_fc/add	FLOAT32	[1, 10]	6	{u'details_type': 0}
14	layer_5_fc/weight/transpose	FLOAT32	[10, 200]	2	{u'details_type': 0}
15	output	FLOAT32	[1, 10]	14	{u'details_type': 0}
16	x	FLOAT32	[1, 28, 28, 1]	16	{u'max': [255.0], u'details_type': 0, u'min': [0.0]}

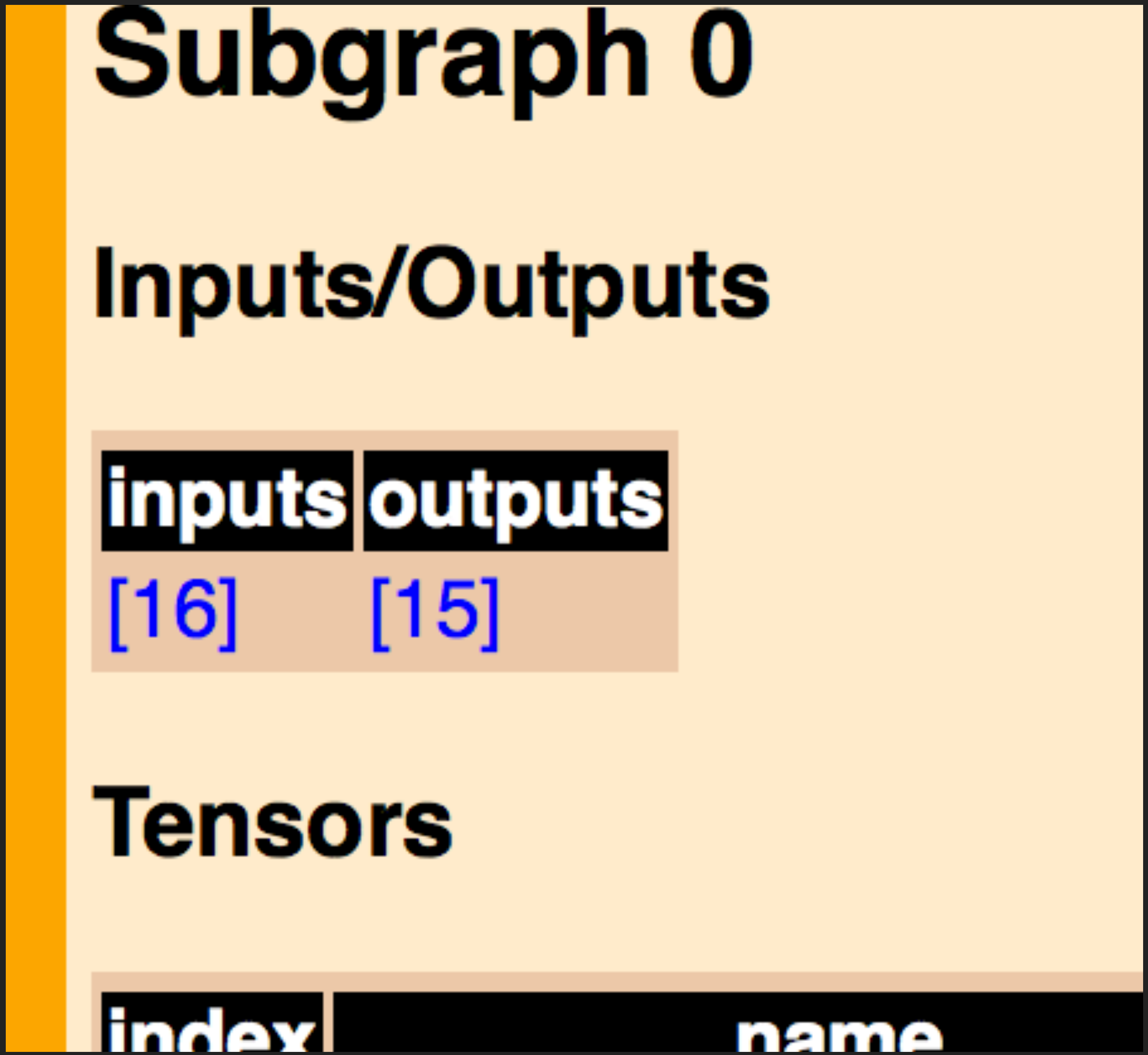
INPUT AND OUTPUT FORMATS IN A TENSOR FLOW LITE MODEL (2)

- inputs: index 16

  ……….

- outputs: index 15

  ……….



Subgraph 0

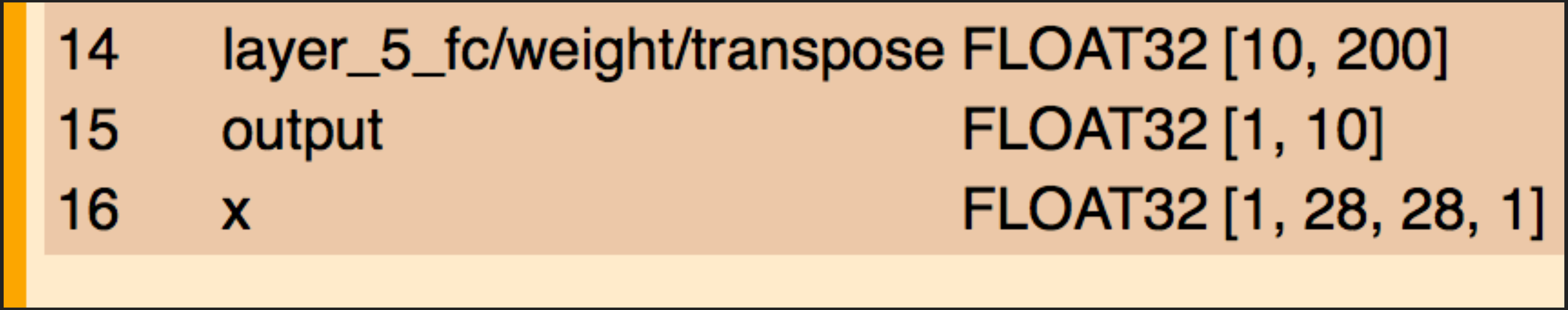
Inputs/Outputs

inputs	outputs
[16]	[15]

Tensors

index	name

INPUT AND OUTPUT FORMATS IN A TENSOR FLOW LITE MODEL (3)



```
14    layer_5_fc/weight/transpose FLOAT32 [10, 200]
15    output                      FLOAT32 [1, 10]
16    x                           FLOAT32 [1, 28, 28, 1]
```

- inputs: index 16
  shape [1, 28, 28, 1]
  type FLOAT32
  name “x”

- outputs: index 15
  shape [1, 10]
  type FLOAT32
  name “output”

LABELLING THE INPUT AND OUTPUT IN THE MODEL CODE

- nex3z/tflite-mnist-android/*train.py*

```
...
def train(mnist_data, options):
 x = tf.placeholder(tf.float32, [None, 28, 28, 1], name='x')
 ...
 with tf.Session() as sess:
  train_writer = tf.summary.FileWriter(options.log_dir + ‘/train',
                                       sess.graph)
  ...
  graph_def = tf.get_default_graph().as_graph_def()
  output_graph = graph_util.convert_variables_to_constants(sess,
                                                           graph_def,
                                                           [‘output'])
  ...
```

INPUT FORMAT FOR MODEL

- Input shape is [1, 28, 28, 1]
- The first number is batch size, so pass in a single image for analysis
- The second and third numbers are the width and height of the image we want to analyze
- The fourth number is the number of values for each pixel; in this case each pixel will be represented by a single FLOAT32 (specified in type). This will represent a greyscale value from 0.0 (black) to 1.0 (white).
- (A bit confusingly, there is also the notion of "index" of the input format, index 0 is used since we have just one input format, not to be confused with the batch size of one)

INPUT FORMAT FOR MODEL (2)

```
...
private val imagePixels = IntArray(imageSize)
...

fun classify(
        bitmap: Bitmap,
        success: (Int, Float, Long) -> Unit
) {
    val inputs = FirebaseModelInputs.Builder()
            .add(bitmap.toVector())
            .build()
    val start = System.currentTimeMillis()
    interpreter.run(inputs, options)
            .addOnSuccessListener { outputs -> ... }
            .addOnFailureListener { exception -> throw(exception) }
}

private fun Bitmap.toVector(): Array<Array<Array<FloatArray>>> {
    getPixels(imagePixels, 0, width, 0, 0, width, height)
    return Array(1) {
        Array(imageHeight) { y ->
            Array(imageWidth) { x ->
                floatArrayOf(imagePixels[x + (y * imageWidth)].convertToGreyScale())
            }
        }
    }
}
```

OUTPUT FORMAT FOR MODEL

- Output shape is [1, 10]
- The first number is the batch size, conforming with the input format batch
- The second number represents the number of classifications. Our model will classify the image from 0-9 inclusive, and each classification will have a FLOAT32 (type) value giving the probability that the input image is that digit classification.

```
[7.906156E-16, 2.7768906E-15, 3.609502E-14, 1.0,
5.1401704E-20, 2.7919297E-12, 7.474837E-17, 6.025316E-17,
1.3942689E-16, 3.370442E-19]
```

- In above array, the fourth classification has max value 1.0, so inferred digit is 3
- (There is also an analogous notion of “index” of the output format, index 0 is used since we also have just one output format.)

OUTPUT FORMAT FOR MODEL (2)

```
fun classify(
        bitmap: Bitmap,
        success: (Int, Float, Long) -> Unit
) {
    val inputs = FirebaseModelInputs.Builder().add(bitmap.toVector()).build()
    val start = System.currentTimeMillis()
    interpreter.run(inputs, options)
            .addOnSuccessListener { outputs -> outputs.map().entries.maxBy { it.value }
                    ?.also { maxEntry ->
                        success(maxEntry.key, maxEntry.value,
                                System.currentTimeMillis() - start)
                    }
            }
            .addOnFailureListener { exception -> throw(exception) }
}
...

private fun FirebaseModelOutputs.map(): Map<Int, Float> {
    return getOutput<Array<FloatArray>>(0)[0]
            .mapIndexed { index, fl -> index to fl }.toMap()
}
```

WHERE IS THE BAD

- It’s all good… but Mark Allison did share one gotcha
  - “I had got everything working but the accuracy of the digit detection was pretty poor.”
- The Bad: Incorrect assumptions about Training Dataset
- The MNIST images were white digits drawn on black b/g; so subtract original greyscale value calculated from 1.0

```
private fun Int.convertToGreyScale(): Float =
        1f - ((Color.red(this) + Color.green(this) + Color.blue(this))
                .toFloat() / 3f / 255f)
```

WRAP UP - ANOTHER EXAMPLE INPUT AND OUTPUT FORMATS

- In TFLG > Android demo app: camera demo app
  - If use: quantized Mobilenet TensorFlow Lite model
  - Input shape is 1 * 224 * 224 * 3
    - 1 image in a batch
    - 224 x 224 width and height of image
    - 3 bytes (type “uint8” see www.tensorflow.org/lite/tf_ops_compatibility#supported_types)
      - for the three colors of a pixel
  - Output shape is 1 * 1001
    - the model has 1001 unique categories for the image

APPENDIX 1 – OVERVIEW OF APP LOGIC FLOW

- In MainActivity, a custom view representing the finger canvas is created. When FingerCanvasView view's onTouchEvent(MotionEvent) is called with ACTION_UP, drawingListener(bitmap) is called
- In MainActivity, the drawingListener(bitmap) listener function is *defined* to launch a coroutine that calls numberClassifier.classify(bitmap, *success*); the *success* lambda is *defined*: 1) to take three parameters: the first parameter would be the inferred digit; 2) to launch another coroutine to update the text view at top of app to display that inferred digit passed in
- The magic happens in the NumberClassifier classify(bitmap, *success*) method. That class has a FirebaseModelInterpreter interpreter. The method first converts the bitmap into FirebaseModelInputs inputs with the expected input format, then calls interpreter.run(inputs, options) where options defines the input and output formats; the run call returns a Task<FirebaseModelOutputs> task on which we call addOnSuccessListener(<lambda>) to add a onSuccess lambda that takes a FirebaseModelOutputs outputs parameter. The onSuccess lambda extracts the output array from the FirebaseModelOutputs outputs since we know the output format, then finds the array element (indexed from 0 to 9) with the max value (which is a probability value), then calls the *success* lambda with the index of the max element as the first parameter

APPENDIX 2 – LINKS TO USEFUL BLOG POSTS, TUTORIALS

- https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41
- https://firebase.google.com/docs/ml-kit/android/use-custom-models