

Mobile App and Image training on device:

1. App Starter Screenshot

The screenshot shows the Android Studio interface. On the left, the code editor displays the `MainActivity.kt` file with Java code for object detection. On the right, the emulator shows the app's initial screen titled "Object Detection Codelab" with the sub-titile "Object detection demo". Below the title, there is a message "Select a preset image or take a new photo" and three thumbnail images of food items. A purple button labeled "Take photo" is at the bottom.

```
130     * runObjectDetection(bitmap: Bitmap)
131     *   TFLite Object Detection function
132     */
133     private fun runObjectDetection(bitmap: Bitmap) {
134         //TODO: Add object detection code here
135         val image = TensorImage.fromBitmap(bitmap)
136
137         // Step 2: Initialize the detector object
138         val options = ObjectDetector.ObjectDetectorOptions.builder()
139             .setMaxResults(5)
140             .setScoreThreshold(0.5f)
141             .build()
142         val detector = ObjectDetector.createFromFileAndOptions(
143             context: this, // the application context
144             modelPath: "salad.tflite", // must be same as the filename in assets folder
145             options
146         )
147
148         // Step 3: feed given image to the model and print the detection result
149         val results = detector.detect(image)
150
151         // Step 4: Parse the detection result and show it
152         val resultToDisplay = results.map { it:Detection!
153             // Get the top-1 category and craft the display text
154             val category = it.categories.first()
155             val text = "${category.label}, ${category.score.times( other: 100).toInt()}%"
156
157             // Create a data object to display the detection result
158             DetectionResult(it.boundingBox, text) ^map
159         }
160
161         // Draw the detection result on the bitmap and show it.
162         val imgWithResult = drawDetectionResult(bitmap, resultToDisplay)
163         runOnUiThread {
164             inputImageView.setImageBitmap(imgWithResult)
165         }
166     }
```

2. Objects Detection

The screenshot shows the Android Studio interface. On the left, the code editor displays the `MainActivity.kt` file with Java code for object detection. On the right, the emulator shows the app's detection results screen titled "Object Detection Codelab". The screen displays a photograph of a meal with various objects detected and labeled with bounding boxes and confidence scores. Labels include "wine glass, 6%", "dining table, 38%", and "bowl, 54%". Below the image, there is a message "Select a preset image or take a new photo" and three thumbnail images of food items. A purple button labeled "Take photo" is at the bottom.

```
130     * runObjectDetection(bitmap: Bitmap)
131     *   TFLite Object Detection function
132     */
133     private fun runObjectDetection(bitmap: Bitmap) {
134         //TODO: Add object detection code here
135         val image = TensorImage.fromBitmap(bitmap)
136
137         // Step 2: Initialize the detector object
138         val options = ObjectDetector.ObjectDetectorOptions.builder()
139             .setMaxResults(5)
140             .setScoreThreshold(0.5f)
141             .build()
142         val detector = ObjectDetector.createFromFileAndOptions(
143             context: this, // the application context
144             modelPath: "model.tflite", // must be same as the filename in assets folder
145             options
146         )
147
148         // Step 3: feed given image to the model and print the detection result
149         val results = detector.detect(image)
150
151         // Step 4: Parse the detection result and show it
152         val resultToDisplay = results.map { it:Detection!
153             // Get the top-1 category and craft the display text
154             val category = it.categories.first()
155             val text = "${category.label}, ${category.score.times( other: 100).toInt()}%"
156
157             // Create a data object to display the detection result
158             DetectionResult(it.boundingBox, text) ^map
159         }
160
161         // Draw the detection result on the bitmap and show it.
162         val imgWithResult = drawDetectionResult(bitmap, resultToDisplay)
163         runOnUiThread {
164             inputImageView.setImageBitmap(imgWithResult)
165         }
166     }
```

3. Custom Model



The screenshot shows the 'Object Detection Codelab' app running on an emulator. The main screen displays a salad image with three detected objects: 'Subj: 56854-Nato, 69%' (highlighted in yellow), 'Cheese, 58%' (highlighted in green), and 'Tomato, 74%' (highlighted in red). Below the image is a button labeled 'Select a preset image or take a new photo' and a camera icon. At the bottom right is a 'Take photo' button.

```

130     * runObjectDetection(bitmap: Bitmap)
131     *      TFLite Object Detection function
132     */
133     private fun runObjectDetection(bitmap: Bitmap) {
134         //TODO: Add object detection code here
135         val image = TensorImage.fromBitmap(bitmap)
136
137         // Step 2: Initialize the detector object
138         val options = ObjectDetector.ObjectDetectorOptions.builder()
139             .setMaxResults(5)
140             .setScoreThreshold(0.5f)
141             .build()
142         val detector = ObjectDetector.createFromFileAndOptions(
143             context, // the application context
144             modelPath: "salad.tflite", // must be same as the filename in assets folder
145             options
146         )
147
148         // Step 3: feed given image to the model and print the detection result
149         val results = detector.detect(image)
150
151         // Step 4: Parse the detection result and show it
152         val resultToDisplay = results.map { it: Detection! }
153             // Get the top-1 category and craft the display text
154             val category = it.categories.first()
155             val text = "${category.label}, ${category.score.times( other: 100).toInt()}%"
156
157             // Create a data object to display the detection result
158             DetectionResult(it.boundingBox, text) ^map
159         }
160
161         // Draw the detection result on the bitmap and show it.
162         val imgWithResult = drawDetectionResult(bitmap, resultToDisplay)
163         runOnUiThread {
164             imageView.setImageBitmap(imgWithResult)
165         }
166     }

```

B. Mobile app and audio training on device : Build a custom pre-trained Audio Classification model



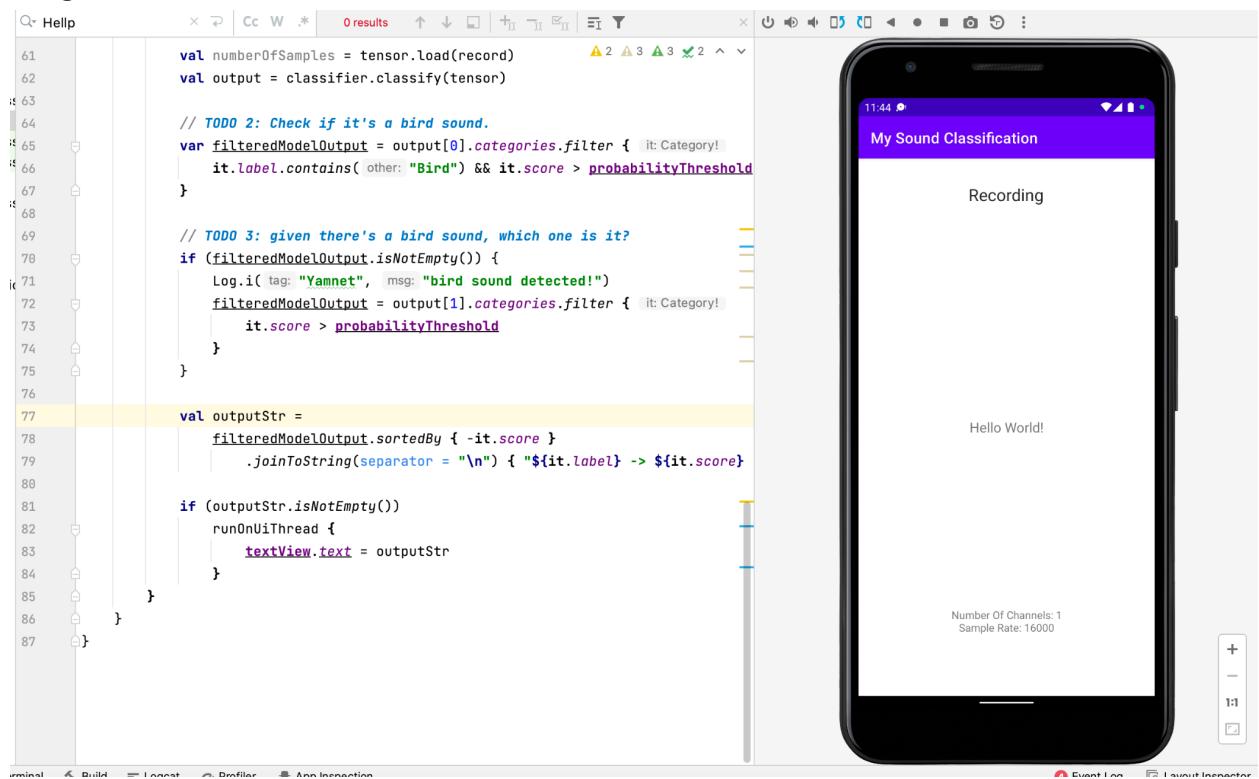
The screenshot shows the 'My Sound Classification' app running on an emulator. The main screen displays the text 'Recording'. Below it, a message says 'Hello World!'. At the bottom, it shows 'Number Of Channels: 1' and 'Sample Rate: 16000'. The code editor on the left shows the following Java code:

```

61     val numberOfSamples = tensor.load(record)
62     val output = classifier.classify(tensor)
63
64     // TODO 2: Check if it's a bird sound.
65     var filteredModelOutput = output[0].categories.filter { it: Category!
66         it.label.contains( other: "Bird") && it.score > probabilityThreshold
67     }
68
69     // TODO 3: given there's a bird sound, which one is it?
70     if (filteredModelOutput.isNotEmpty()) {
71         Log.i( tag: "Yamnet", msg: "bird sound detected!")
72         filteredModelOutput = output[1].categories.filter { it: Category!
73             it.score > probabilityThreshold
74         }
75     }
76
77     val outputStr =
78         filteredModelOutput.sortedBy { -it.score }
79             .joinToString(separator = "\n") { "${it.label} -> ${it.score}"
80
81     if (outputStr.isNotEmpty())
82         runOnUiThread {
83             textView.text = outputStr
84         }
85     }
86
87 }

```

Image 2 with audio classification



The screenshot shows the Android Studio interface. On the left is the Java code editor with code for audio classification. On the right is a virtual device displaying the app's user interface.

Java Code (audioClassification.java):

```
61     val numberOfSamples = tensor.load(record)
62     val output = classifier.classify(tensor)
63
64     // TODO 2: Check if it's a bird sound.
65     var filteredModelOutput = output[0].categories.filter { it: Category!
66         it.label.contains( other: "Bird") && it.score > probabilityThreshold
67     }
68
69     // TODO 3: given there's a bird sound, which one is it?
70     if (filteredModelOutput.isNotEmpty()) {
71         Log.i( tag: "Yamnet", msg: "bird sound detected!")
72         filteredModelOutput = output[1].categories.filter { it: Category!
73             it.score > probabilityThreshold
74         }
75     }
76
77     val outputStr =
78         filteredModelOutput.sortedBy { -it.score }
79         .joinToString(separator = "\n") { "${it.label} -> ${it.score}"}
80
81     if (outputStr.isNotEmpty())
82         runOnUiThread {
83             textView.text = outputStr
84         }
85     }
86 }
87 }
```

App Screenshot:

The app's title bar says "My Sound Classification". The main screen displays "Recording" at the top. Below it, the text "Hello World!" is shown. At the bottom, it says "Number Of Channels: 1" and "Sample Rate: 16000".

C. Web app and image training on device: TensorFlow.js Transfer Learning Image Classifier. Image 1 about class 1 prediction

codepen.io/jasonmayes/pen/BaNjLyo

Tensorflow.js Boilerplate

Jason Mayes PRO + Follow

HTML

```
<button class="dataCollector" data-lhot="0" data-name="Class 1">Gather Class 1 Data</button>
<button class="dataCollector" data-lhot="1" data-name="Class 2">Gather Class 2 Data</button>
```

JS

```
function reset() {
```

CSS

```
display: block,
margin: 10px;
background: #000000;
width: 640px;
height: 480px;
```

Talking: Vijay Eranti

Prediction: Class 1 with 99% confidence



Gather Class 1 Data Gather Class 2 Data Train & Predict! Reset

Console Assets Comments % Keys Fork Embed Export Share

Image 2 about class 2 prediction

codepen.io/jasonmayes/pen/BaNjLyo

Tensorflow.js Boilerplate

Jason Mayes PRO + Follow

HTML

```
<button class="dataCollector" data-lhot="0" data-name="Class 1">Gather Class 1 Data</button>
<button class="dataCollector" data-lhot="1" data-name="Class 2">Gather Class 2 Data</button>
```

JS

```
function reset() {
```

CSS

```
display: block,
margin: 10px;
background: #000000;
width: 640px;
height: 480px;
```

Talking: Vijay Eranti

Prediction: Class 2 with 99% confidence



Gather Class 1 Data Gather Class 2 Data Train & Predict! Reset

Console Assets Comments % Keys Fork Embed Export Share

D. Audio recognition using transfer learning

1. It will be on left side for one type of sound

The screenshot shows a split-screen interface. On the left is an HTML editor with the following code:

```
models/speech-commands"></script>
5 </head>
6 <body>
7 <button id="left" onmousedown="collect(0)">Left</button>
8 <button id="right" onmousedown="collect(1)">Right</button>
9 <button id="noise" onmousedown="collect(2)">Noise</button>
10 <br/><br/>
11 <button id="train" onclick="train()">Train</button>
12 <br/><br/>
13 <button id="listen" onclick="listen()">Listen</button>
14 <input type="range" id="output" min="0" max="10" step="0.1">
15 <div id="console"></div>
```

On the right is a JS editor with the following code:

```
model = tf.sequential();
model.add(tf.layers.depthwiseConv2d({
  depthMultiplier: 8,
  kernelSize: [NUM_FRAMES, 3],
  activation: 'relu',
  inputShape: INPUT_SHAPE
}));
model.add(tf.layers.maxPooling2d({poolSize: [1, 2], strides: [2, 2]}));
model.add(tf.layers.flatten());
model.add(tf.layers.dense({units: 3, activation: 'softmax'}));
const optimizer = tf.train.adam(0.01);
model.compile({
  optimizer,
  loss: 'categoricalCrossentropy',
  metrics: ['accuracy']});
```

Below the editors are three buttons: Left, Right, and Noise. A "Train" button is also present. At the bottom is a slider labeled "Stop" with its value set to 0.

2. For second sound the slider moves towards right

This screenshot is identical to the first one, except the slider has been moved to the right, with its value now set to 1.

E. Recognize text and facial features with ML Kit:Android

1. Starter App

The screenshot shows the Android Studio interface with the code editor open to `MainActivity.java`. The code is part of a class named `FaceDetection` and contains logic for running text recognition. The `runTextRecognition()` method is highlighted. The code uses `InputImage` and `TextRecognizer` from the ML Kit library to process an image and detect text. The `Emulator` window shows a smartphone displaying the "ML Kit Codelab" app. The app's UI includes a spinner for selecting images, a button labeled "FIND TEXT", and a button labeled "FIND FACE CONTOUR". The background of the app shows a park scene with a sign that reads "Please Walk on the Grass".

```
109     }
110 }
111 });
112 Spinner dropdown = findViewById(R.id.spinner);
113 String[] items = new String[]{"Test Image 1 (Text)", "Test Image 2 (Face)"};
114 ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.
115     .simple_spinner_dropdown_item, items);
116 dropdown.setAdapter(adapter);
117 dropdown.setOnItemSelectedListener(this);
118 }

119 private void runTextRecognition() {
120     InputImage image = InputImage.fromBitmap(mSelectedImage, i: 0);
121     TextRecognizer recognizer = TextRecognition.getClient();
122     mTextButton.setEnabled(false);
123     recognizer.process(image)
124         .addOnSuccessListener(
125             new OnSuccessListener<Text>() {
126                 @Override
127                 public void onSuccess(Text texts) {
128                     mTextButton.setEnabled(true);
129                     processTextRecognitionResult(texts);
130                 }
131             })
132         .addOnFailureListener(
133             new OnFailureListener() {
134                 @Override
135                 public void onFailure(@NonNull Exception e) {
136                     // Task failed with an exception
137                     mTextButton.setEnabled(true);
138                     e.printStackTrace();
139                 }
140             });
141 }
```

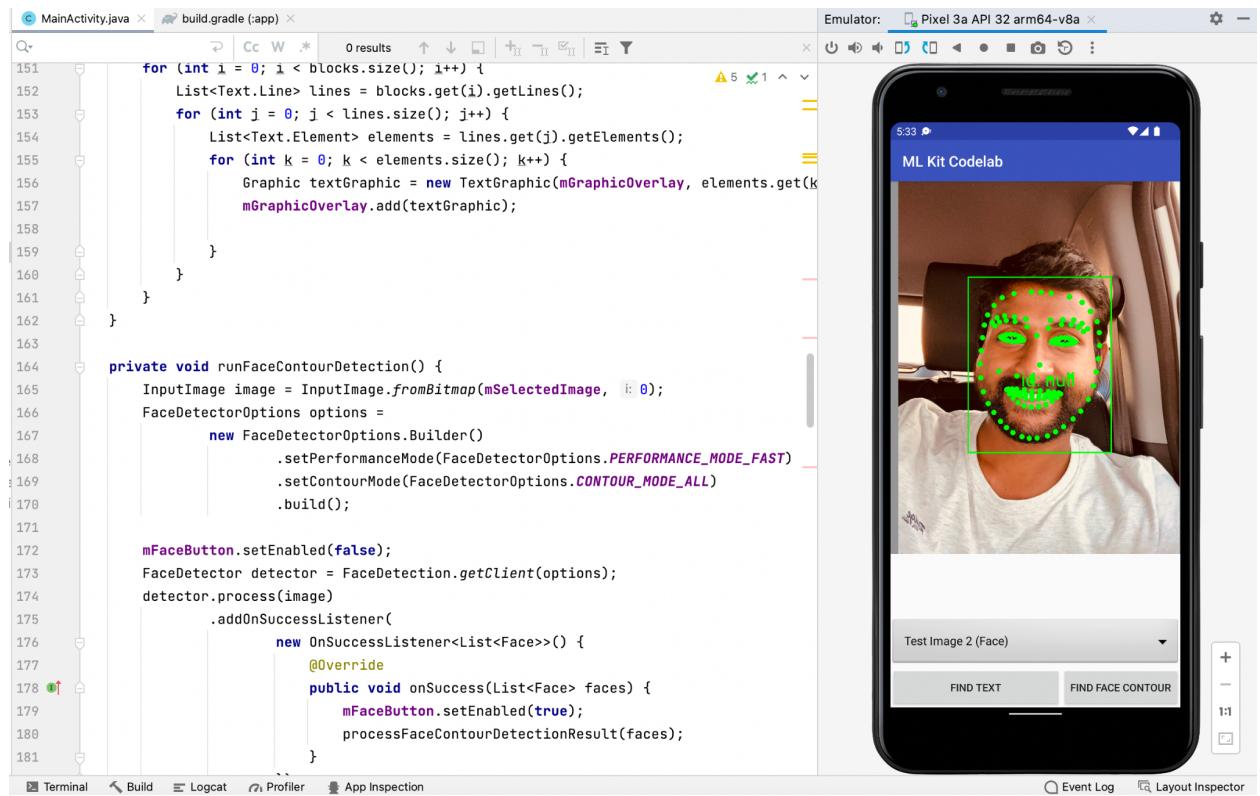
2. Text recognition

This screenshot is identical to the one above, showing the same code in `MainActivity.java` and the same state of the "ML Kit Codelab" app in the emulator. The code in the `runTextRecognition()` method is highlighted, and the emulator shows the app's UI with the "FIND TEXT" button being processed.

```
109     }
110 }
111 });
112 Spinner dropdown = findViewById(R.id.spinner);
113 String[] items = new String[]{"Test Image 1 (Text)", "Test Image 2 (Face)"};
114 ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.
115     .simple_spinner_dropdown_item, items);
116 dropdown.setAdapter(adapter);
117 dropdown.setOnItemSelectedListener(this);
118 }

119 private void runTextRecognition() {
120     InputImage image = InputImage.fromBitmap(mSelectedImage, i: 0);
121     TextRecognizer recognizer = TextRecognition.getClient();
122     mTextButton.setEnabled(false);
123     recognizer.process(image)
124         .addOnSuccessListener(
125             new OnSuccessListener<Text>() {
126                 @Override
127                 public void onSuccess(Text texts) {
128                     mTextButton.setEnabled(true);
129                     processTextRecognitionResult(texts);
130                 }
131             })
132         .addOnFailureListener(
133             new OnFailureListener() {
134                 @Override
135                 public void onFailure(@NonNull Exception e) {
136                     // Task failed with an exception
137                     mTextButton.setEnabled(true);
138                     e.printStackTrace();
139                 }
140             });
141 }
```

3. Face Recognition



The screenshot shows the Android Studio interface with the following components:

- Left Panel (Code Editor):** Displays the `MainActivity.java` file. The code implements logic for processing text blocks and performing face detection. It includes a loop to handle text blocks, another loop to handle lines within each block, and a nested loop to handle elements within each line. It uses `TextGraphic` and `FaceDetectorOptions` to add text overlays and detect faces respectively.
- Right Panel (Emulator):** Shows a Pixel 3a API 32 arm64-v8a emulator running the ML Kit Codelab app. The app interface includes a title bar "ML Kit Codelab", a camera preview area showing a man's face with green bounding boxes and dots indicating detected features, and a bottom toolbar with buttons for "Test Image 2 (Face)", "FIND TEXT", and "FIND FACE CONTOUR".
- Bottom Navigation Bar:** Includes tabs for Terminal, Build, Logcat, Profiler, App Inspection, Event Log, and Layout Inspector.