PRACTICAL: 14

AIM: Create an application that uses the end-to-end process of training a machine learning model that can recognize handwritten digit images with TensorFlow and deploy it to an Android app.

Source Code:
Layout File/s:
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

  <com.divyanshu.draw.widget.DrawView
      android:id="@+id/draw_view"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      app:layout_constraintDimensionRatio="1:1"
      app:layout_constraintTop_toTopOf="parent"/>

  <TextView
      android:id="@+id/predicted_text"
      android:textStyle="bold"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/prediction_text_placeholder"
      android:textSize="20sp"
      app:layout_constraintBottom_toTopOf="@id/clear_button"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toBottomOf="@id/draw_view"/>

  <Button
      android:id="@+id/clear_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/clear_button_text"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Kotlin File/s:
MainActivity.kt

```
package org.tensorflow.lite.codelabs.digitclassifier

import android.annotation.SuppressLint
import android.graphics.Color
import android.os.Bundle
import android.util.Log
import android.view.MotionEvent
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import com.divyanshu.draw.widget.DrawView

class MainActivity : AppCompatActivity() {

  private var drawView: DrawView? = null
  private var clearButton: Button? = null
  private var predictedTextView: TextView? = null
  private var digitClassifier = DigitClassifier(this)

  @SuppressLint("ClickableViewAccessibility")
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Setup view instances.
    drawView = findViewById(R.id.draw_view)
    drawView?.setStrokeWidth(70.0f)
    drawView?.setColor(Color.WHITE)
    drawView?.setBackgroundColor(Color.BLACK)
    clearButton = findViewById(R.id.clear_button)
    predictedTextView = findViewById(R.id.predicted_text)

    // Setup clear drawing button.
    clearButton?.setOnClickListener {
      drawView?.clearCanvas()
      predictedTextView?.text = getString(R.string.prediction_text_placeholder)
    }

    // Setup classification trigger so that it classify after every stroke drew.
    drawView?.setOnTouchListener { _, event ->
      // As we have interrupted DrawView's touch event,
      // we first need to pass touch events through to the instance for the drawing to show up.
      drawView?.onTouchEvent(event)

      // Then if user finished a touch event, run classification
```

```
      if (event.action == MotionEvent.ACTION_UP) {
        classifyDrawing()
      }

      true
    }

    // Setup digit classifier.
    digitClassifier
      .initialize()
      .addOnFailureListener { e -> Log.e(TAG, "Error to setting up digit classifier.", e) }
  }

  override fun onDestroy() {
    // Sync DigitClassifier instance lifecycle with MainActivity lifecycle,
    // and free up resources (e.g. TF Lite instance) once the activity is destroyed.
    digitClassifier.close()
    super.onDestroy()
  }

  private fun classifyDrawing() {
    val bitmap = drawView?.getBitmap()

    if ((bitmap != null) && (digitClassifier.isInitialized)) {
      digitClassifier
        .classifyAsync(bitmap)
        .addOnSuccessListener { resultText -> predictedTextView?.text = resultText }
        .addOnFailureListener { e ->
          predictedTextView?.text = getString(
            R.string.classification_error_message,
            e.localizedMessage
          )
          Log.e(TAG, "Error classifying drawing.", e)
        }
    }
  }

  companion object {
    private const val TAG = "MainActivity"
  }
}
```

Digitclassifier.kt

```
package org.tensorflow.lite.codelabs.digitclassifier

import android.content.Context
import android.content.res.AssetManager
import android.graphics.Bitmap
import android.util.Log
```

```
import com.google.android.gms.tasks.Task
import com.google.android.gms.tasks.Tasks.call
import org.tensorflow.lite.Interpreter
import java.io.FileInputStream
import java.io.IOException
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.nio.channels.FileChannel
import java.util.concurrent.Callable
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors


class DigitClassifier(private val context: Context) {
  // TODO: Add a TF Lite interpreter as a field.
  private var interpreter: Interpreter? = null
  var isInitialized = false
  private set

        /** Executor to run inference task in the background. */
        private val executorService: ExecutorService = Executors.newCachedThreadPool()

  private var inputImageWidth: Int = 0 // will be inferred from TF Lite model.
  private var inputImageHeight: Int = 0 // will be inferred from TF Lite model.
  private var modelInputSize: Int = 0 // will be inferred from TF Lite model.

  fun initialize(): Task<Void> {
    return call(
      executorService,
      Callable<Void> {
        initializeInterpreter()
        null
      }
    )
  }

  @Throws(IOException::class)
  private fun initializeInterpreter() {
    // TODO: Load the TF Lite model from file and initialize an interpreter.
    val assetManager = context.assets
    val model = loadModelFile(assetManager, "mnist.tflite")

    // Initialize TF Lite Interpreter with NNAPI enabled.
    val options = Interpreter.Options()
    options.setUseNNAPI(true)
    val interpreter = Interpreter(model, options)

    // TODO: Read the model input shape from model file.
    val inputShape = interpreter.getInputTensor(0).shape()
    inputImageWidth = inputShape[1]
    inputImageHeight = inputShape[2]
```

```
    modelInputSize = FLOAT_TYPE_SIZE * inputImageWidth * inputImageHeight * PIXEL_SIZE

    this.interpreter = interpreter

    isInitialized = true
    Log.d(TAG, "Initialized TFLite interpreter.")
  }

  @Throws(IOException::class)
  private fun loadModelFile(assetManager: AssetManager, filename: String): ByteBuffer {
    val fileDescriptor = assetManager.openFd(filename)
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
    val fileChannel = inputStream.channel
    val startOffset = fileDescriptor.startOffset
    val declaredLength = fileDescriptor.declaredLength
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength)
  }

  private fun classify(bitmap: Bitmap): String {
    check(isInitialized) { "TF Lite Interpreter is not initialized yet." }

    // TODO: Add code to run inference with TF Lite.
// Preprocessing: resize the input image to match the model input shape.
    val resizedImage = Bitmap.createScaledBitmap(
      bitmap,
      inputImageWidth,
      inputImageHeight,
      true
    )
    val byteBuffer = convertBitmapToByteBuffer(resizedImage)

    // Define an array to store the model output.
    val output = Array(1) { FloatArray(OUTPUT_CLASSES_COUNT) }

// Run inference with the input data.
    interpreter?.run(byteBuffer, output)
    // Post-processing: find the digit that has the highest probability
// and return it a human-readable string.
    val result = output[0]
    val maxIndex = result.indices.maxBy { result[it] } ?: -1
    val resultString = "Prediction Result: %d\nConfidence: %2f".
      format(maxIndex, result[maxIndex])

    return resultString
  }

  fun classifyAsync(bitmap: Bitmap): Task<String> {
    return call(executorService, Callable<String> { classify(bitmap) })
  }

  fun close() {
```

```
    call(
      executorService,
      Callable<String> {
        // TODO: close the TF Lite interpreter here
        interpreter?.close()

        Log.d(TAG, "Closed TFLite interpreter.")
        null
      }
    )
  }

  private fun convertBitmapToByteBuffer(bitmap: Bitmap): ByteBuffer {
    val byteBuffer = ByteBuffer.allocateDirect(modelInputSize)
    byteBuffer.order(ByteOrder.nativeOrder())

    val pixels = IntArray(inputImageWidth * inputImageHeight)
    bitmap.getPixels(pixels, 0, bitmap.width, 0, 0, bitmap.width, bitmap.height)

    for (pixelValue in pixels) {
      val r = (pixelValue shr 16 and 0xFF)
      val g = (pixelValue shr 8 and 0xFF)
      val b = (pixelValue and 0xFF)

      // Convert RGB to grayscale and normalize pixel value to [0..1].
      val normalizedPixelValue = (r + g + b) / 3.0f / 255.0f
      byteBuffer.putFloat(normalizedPixelValue)
    }

    return byteBuffer
  }

  companion object {
    private const val TAG = "DigitClassifier"

    private const val FLOAT_TYPE_SIZE = 4
    private const val PIXEL_SIZE = 1

    private const val OUTPUT_CLASSES_COUNT = 10
  }
}
```

Output:


2:29
WCMC_PR14_17IT104
Prediction Result: 8
Confidence: 12.172560
CLEAR


2:29
WCMC_PR14_17IT104
Prediction Result: 4
Confidence: 8.298666
CLEAR