Files in android application AslTfLite:

CameraActivity.kt

```
/*
 * https://github.com/android/camera-samples/tree/main/CameraXTfLite
 * Copyright 2020 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *    https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.example.asltflite

import android.Manifest
import android.annotation.SuppressLint
import android.content.Context
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.Matrix
import android.graphics.RectF
import android.os.Bundle
import android.util.Log
import android.util.Size
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.AspectRatio
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.lifecycle.LifecycleOwner
import com.example.asltflite.databinding.ActivityCameraBinding
import kotlinx.android.synthetic.main.activity_camera.*
import org.tensorflow.lite.DataType
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.nnapi.NnApiDelegate
import org.tensorflow.lite.support.common.FileUtil
import org.tensorflow.lite.support.common.ops.NormalizeOp
```

```
import org.tensorflow.lite.support.image.ImageProcessor
import org.tensorflow.lite.support.image.TensorImage
import org.tensorflow.lite.support.image.ops.ResizeOp
import org.tensorflow.lite.support.image.ops.ResizeWithCropOrPadOp
import org.tensorflow.lite.support.image.ops.Rot90Op
import java.util.concurrent.Executors
import kotlin.math.min
import kotlin.random.Random

/** Activity that displays the camera and performs object detection on the incoming frames */
class CameraActivity : AppCompatActivity() {

  private lateinit var activityCameraBinding: ActivityCameraBinding

  private lateinit var bitmapBuffer: Bitmap

  private val executor = Executors.newSingleThreadExecutor()
  private val permissions = listOf(Manifest.permission.CAMERA)
  private val permissionsRequestCode = Random.nextInt(0, 10000)

  private var lensFacing: Int = CameraSelector.LENS_FACING_BACK
  private val isFrontFacing get() = lensFacing == CameraSelector.LENS_FACING_FRONT

  private var pauseAnalysis = false
  private var imageRotationDegrees: Int = 0
  private val tfImageBuffer = TensorImage(DataType.FLOAT32)

  private val tfImageProcessor by lazy {
    val cropSize = minOf(bitmapBuffer.width, bitmapBuffer.height)
    ImageProcessor.Builder()
      .add(ResizeWithCropOrPadOp(cropSize, cropSize))
      .add(ResizeOp(
        tfInputSize.height, tfInputSize.width, ResizeOp.ResizeMethod.NEAREST_NEIGHBOR)
      )
      .add(Rot90Op(-imageRotationDegrees / 90))
      .add(NormalizeOp(0f, 1f))
      .build()
  }

  // nnAPiDelegate must be released by explicitly calling its close() function.
  //    https://github.com/android/camera-samples/issues/417
  private var nnapiInitialized = false
  private val nnApiDelegate by lazy  {
    NnApiDelegate().apply { nnapiInitialized = true }
  }

  private val tflite by lazy {
    Interpreter(
      FileUtil.loadMappedFile(this, MODEL_PATH),
```

```
        Interpreter.Options().addDelegate(nnApiDelegate))
}

private val detector by lazy {
    ObjectDetectionHelper(
        tflite,
        FileUtil.loadLabels(this, LABELS_PATH)
    )
}

private val tfInputSize by lazy {
    val inputIndex = 0
    val inputShape = tflite.getInputTensor(inputIndex).shape()
    Size(inputShape[2], inputShape[1]) // Order of axis is: {1, height, width, 3}
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    activityCameraBinding = ActivityCameraBinding.inflate(layoutInflater)
    setContentView(activityCameraBinding.root)

    activityCameraBinding.cameraCaptureButton.setOnClickListener {

        // Disable all camera controls
        it.isEnabled = false

        if (pauseAnalysis) {
            // If image analysis is in paused state, resume it
            pauseAnalysis = false
            activityCameraBinding.imagePredicted.visibility = View.GONE

        } else {
            // Otherwise, pause image analysis and freeze image
            pauseAnalysis = true
            val matrix = Matrix().apply {
                postRotate(imageRotationDegrees.toFloat())
                if (isFrontFacing) postScale(-1f, 1f)
            }
            val uprightImage = Bitmap.createBitmap(
                bitmapBuffer, 0, 0, bitmapBuffer.width, bitmapBuffer.height, matrix, true)
            activityCameraBinding.imagePredicted.setImageBitmap(uprightImage)
            activityCameraBinding.imagePredicted.visibility = View.VISIBLE
        }

        // Re-enable camera controls
        it.isEnabled = true
    }
}
```

```
override fun onDestroy() {
    if (nnapiInitialized) nnApiDelegate.close()
    super.onDestroy()
}

/** Declare and bind preview and analysis use cases */
@SuppressLint("UnsafeExperimentalUsageError")
private fun bindCameraUseCases() = activityCameraBinding.viewFinder.post {

    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
    cameraProviderFuture.addListener(Runnable {

        // Camera provider is now guaranteed to be available
        val cameraProvider = cameraProviderFuture.get()

        // Set up the view finder use case to display camera preview
        val preview = Preview.Builder()
            .setTargetAspectRatio(AspectRatio.RATIO_4_3)
            .setTargetRotation(activityCameraBinding.viewFinder.display.rotation)
            .build()

        // Set up the image analysis use case which will process frames in real time
        val imageAnalysis = ImageAnalysis.Builder()
            .setTargetAspectRatio(AspectRatio.RATIO_4_3)
            .setTargetRotation(activityCameraBinding.viewFinder.display.rotation)
            .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
            .setOutputImageFormat(ImageAnalysis.OUTPUT_IMAGE_FORMAT_RGBA_8888)
            .build()

        var frameCounter = 0
        var lastFpsTimestamp = System.currentTimeMillis()

        imageAnalysis.setAnalyzer(executor, ImageAnalysis.Analyzer { image ->
            if (!::bitmapBuffer.isInitialized) {
                // The image rotation and RGB image buffer are initialized only once
                // the analyzer has started running
                imageRotationDegrees = image.imageInfo.rotationDegrees
                bitmapBuffer = Bitmap.createBitmap(
                    image.width, image.height, Bitmap.Config.ARGB_8888)
            }

            // Early exit: image analysis is in paused state
            if (pauseAnalysis) {
                image.close()
                return@Analyzer
            }

            // Copy out RGB bits to our shared buffer
            image.use { bitmapBuffer.copyPixelsFromBuffer(image.planes[0].buffer)  }
```

```
            // Process the image in Tensorflow
            val tfImage =  tfImageProcessor.process(tfImageBuffer.apply { load(bitmapBuffer) })

            // Perform the object detection for the current frame
            val predictions = detector.predict2(tfImage)
            val labelMap = FileUtil.loadLabels(this, LABELS_PATH)
            val maxIndex = predictions.indexOf(predictions.maxByOrNull { it.score })
            val maxLabel = labelMap[maxIndex]
            // Report only the top prediction
            reportPrediction(predictions.maxByOrNull { it.score }, maxLabel)

            // Compute the FPS of the entire pipeline
            val frameCount = 10
            if (++frameCounter % frameCount == 0) {
              frameCounter = 0
              val now = System.currentTimeMillis()
              val delta = now - lastFpsTimestamp
              val fps = 1000 * frameCount.toFloat() / delta
              Log.d(TAG, "FPS: ${"%.02f".format(fps)} with tensorSize: ${tfImage.width} x ${tfImage.height},
$predictions")
              lastFpsTimestamp = now
            }
          })

          // Create a new camera selector each time, enforcing lens facing
          val cameraSelector = CameraSelector.Builder().requireLensFacing(lensFacing).build()

          // Apply declared configs to CameraX using the same lifecycle owner
          cameraProvider.unbindAll()
          val camera = cameraProvider.bindToLifecycle(
            this as LifecycleOwner, cameraSelector, preview, imageAnalysis)

          // Use the camera object to link our preview use case with the view
          preview.setSurfaceProvider(activityCameraBinding.viewFinder.surfaceProvider)

        }, ContextCompat.getMainExecutor(this))
    }

    private fun reportPrediction(
        prediction: ObjectDetectionHelper.ObjectPrediction2?,  label: String
    ) = activityCameraBinding.viewFinder.post {

        // Early exit: if prediction is not good enough, don't report it
        if (prediction == null || prediction.score < ACCURACY_THRESHOLD) {
            activityCameraBinding.boxPrediction.visibility = View.GONE
            activityCameraBinding.textPrediction.visibility = View.GONE
            return@post
        }
```

```
    // Location has to be mapped to our local coordinates
    //val location = mapOutputCoordinates(prediction.location)

    // Update the text and UI
    activityCameraBinding.textPrediction.text = "${"%.2f".format(prediction.score)} ${label}"
    (activityCameraBinding.boxPrediction.layoutParams as ViewGroup.MarginLayoutParams).apply {
     //   topMargin = location.top.toInt()
     //   leftMargin = location.left.toInt()
     //   width = min(activityCameraBinding.viewFinder.width, location.right.toInt() - location.left.toInt())
      //  height = min(activityCameraBinding.viewFinder.height, location.bottom.toInt() - location.top.toInt())
        topMargin = activityCameraBinding.viewFinder.height / 4
        leftMargin = activityCameraBinding.viewFinder.width / 8
        width = activityCameraBinding.viewFinder.width / 4 * 3
        height = activityCameraBinding.viewFinder.height / 3
    }

    // Make sure all UI elements are visible
    activityCameraBinding.boxPrediction.visibility = View.VISIBLE
    activityCameraBinding.textPrediction.visibility = View.VISIBLE
}

/**
 * Helper function used to map the coordinates for objects coming out of
 * the model into the coordinates that the user sees on the screen.
 */
private fun mapOutputCoordinates(location: RectF): RectF {

    // Step 1: map location to the preview coordinates
    val previewLocation = RectF(
        location.left * activityCameraBinding.viewFinder.width,
        location.top * activityCameraBinding.viewFinder.height,
        location.right * activityCameraBinding.viewFinder.width,
        location.bottom * activityCameraBinding.viewFinder.height
    )

    // Step 2: compensate for camera sensor orientation and mirroring
    val isFrontFacing = lensFacing == CameraSelector.LENS_FACING_FRONT
    val correctedLocation = if (isFrontFacing) {
        RectF(
            activityCameraBinding.viewFinder.width - previewLocation.right,
            previewLocation.top,
            activityCameraBinding.viewFinder.width - previewLocation.left,
            previewLocation.bottom)
    } else {
        previewLocation
    }

    // Step 3: compensate for 1:1 to 4:3 aspect ratio conversion + small margin
```

```
    val margin = 0.1f
    val requestedRatio = 4f / 3f
    val midX = (correctedLocation.left + correctedLocation.right) / 2f
    val midY = (correctedLocation.top + correctedLocation.bottom) / 2f
    return if (activityCameraBinding.viewFinder.width < activityCameraBinding.viewFinder.height) {
        RectF(
            midX - (1f + margin) * requestedRatio * correctedLocation.width() / 2f,
            midY - (1f - margin) * correctedLocation.height() / 2f,
            midX + (1f + margin) * requestedRatio * correctedLocation.width() / 2f,
            midY + (1f - margin) * correctedLocation.height() / 2f
        )
    } else {
        RectF(
            midX - (1f - margin) * correctedLocation.width() / 2f,
            midY - (1f + margin) * requestedRatio * correctedLocation.height() / 2f,
            midX + (1f - margin) * correctedLocation.width() / 2f,
            midY + (1f + margin) * requestedRatio * correctedLocation.height() / 2f
        )
    }
}

override fun onResume() {
    super.onResume()

    // Request permissions each time the app resumes, since they can be revoked at any time
    if (!hasPermissions(this)) {
        ActivityCompat.requestPermissions(
            this, permissions.toTypedArray(), permissionsRequestCode)
    } else {
        bindCameraUseCases()
    }
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == permissionsRequestCode && hasPermissions(this)) {
        bindCameraUseCases()
    } else {
        finish() // If we don't have the required permissions, we can't run
    }
}

/** Convenience method used to check if all permissions required by this app are granted */
private fun hasPermissions(context: Context) = permissions.all {
    ContextCompat.checkSelfPermission(context, it) == PackageManager.PERMISSION_GRANTED
```

```
    }

    companion object {
        private val TAG = CameraActivity::class.java.simpleName

        private const val ACCURACY_THRESHOLD = 0.2f
        private const val MODEL_PATH = "0.9365Val_accuracy_COLOR.tflite"
        private const val LABELS_PATH = "labels.txt"
    }
}
```

ObjectDetectionHelper.kt

```
/*
 * https://github.com/android/camera-samples/tree/main/CameraXTfLite
 * Copyright 2020 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.example.asltflite

import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.support.image.TensorImage
import org.tensorflow.lite.support.label.TensorLabel

/**
 * Helper class used to communicate between our app and the TF object detection model
 */
class ObjectDetectionHelper(private val tflite: Interpreter, private val labels: List<String>) {

    /** Abstraction object that wraps a prediction output in an easy to parse way */
    //data class ObjectPrediction(val location: RectF, val label: String, val score: Float)
    data class ObjectPrediction2(val label: String, val score: Float)

    //private val locations = arrayOf(Array(OBJECT_COUNT) { FloatArray(4) })
    private val labelIndices = arrayOf(FloatArray(OBJECT_COUNT))
    private val scores = arrayOf(FloatArray(OBJECT_COUNT))

    // The original example included view box location data and an output buffer that was not
```

```
    // easily compatible with our .tflite model output
    /**
    private val outputBuffer = mapOf(
        0 to locations,
        1 to labelIndices,
        2 to scores,
        3 to FloatArray(1)
    ) */

    val outputBuffer2 = mapOf(
        0 to scores
    )

    val predictions2 get() = (0 until OBJECT_COUNT - 1).map {
            ObjectPrediction2(
                label = labels[labelIndices[0][it].toInt()],
                score = scores[0][it]
            )

    }

    fun predict2(image: TensorImage): List<ObjectPrediction2> {
        tflite.runForMultipleInputsOutputs(arrayOf(image.buffer), outputBuffer2)
        return predictions2
    }
    companion object {
        const val OBJECT_COUNT = 28
    }

}
```

activity_camera.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ https://github.com/android/camera-samples/tree/main/CameraXTfLite
  ~ Copyright 2019 Google LLC
  ~
  ~ Licensed under the Apache License, Version 2.0 (the "License");
  ~ you may not use this file except in compliance with the License.
  ~ You may obtain a copy of the License at
  ~
  ~     https://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
  -->
```

```
<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:id="@+id/camera_container"
    android:background="@android:color/black"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.camera.view.PreviewView
        android:id="@+id/view_finder"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ImageView
        android:id="@+id/image_predicted"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:visibility="gone" />

    <TextView
        android:id="@+id/text_prediction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/margin_xsmall"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:textAllCaps="true"
        android:textAppearance="@style/TextAppearance.AppCompat.Display1"
        android:text="@string/unknown" />

    <View
        android:id="@+id/box_prediction"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:background="@drawable/shape_rectangle"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <!-- Camera control buttons -->

    <ImageButton
        android:id="@+id/camera_capture_button"
```

```
        android:layout_width="@dimen/round_button_large"
        android:layout_height="@dimen/round_button_large"
        android:layout_marginBottom="@dimen/shutter_button_margin"
        android:scaleType="fitCenter"
        android:background="@drawable/ic_shutter"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:contentDescription="@string/capture_button_alt"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
   https://github.com/android/camera-samples/tree/main/CameraXTfLite
  ~ Copyright 2020 Google LLC
  ~
  ~ Licensed under the Apache License, Version 2.0 (the "License");
  ~ you may not use this file except in compliance with the License.
  ~ You may obtain a copy of the License at
  ~
  ~     https://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
  -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.asltflite">
    <uses-feature android:name="android.hardware.camera.any"/>

    <uses-permission android:name="android.permission.CAMERA"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        tools:ignore="AllowBackup,GoogleAppIndexingWarning">

        <activity
            android:name="com.example.asltflite.CameraActivity"
            android:rotationAnimation="seamless"
            tools:targetApi="O"
```

```
        android:exported="true">
        <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
      </activity>
    </application>

</manifest>
```

build.gradle

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}

android {
    compileSdk 31

    defaultConfig {
        applicationId "com.example.asltflite"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    aaptOptions {
        noCompress "tflite"
        noCompress "lite"
    }
    buildFeatures {
        viewBinding true
```

```
        mlModelBinding true
    }
}

dependencies {
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'

    // CameraX
    def camerax_version = "1.1.0-alpha09"
    implementation "androidx.camera:camera-core:${camerax_version}"
    implementation "androidx.camera:camera-camera2:${camerax_version}"
    implementation "androidx.camera:camera-lifecycle:${camerax_version}"

    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    implementation 'androidx.exifinterface:exifinterface:1.3.3'
    implementation "androidx.camera:camera-view:1.0.0-alpha29"
    implementation 'com.google.android.material:material:1.4.0'

    implementation 'org.tensorflow:tensorflow-lite-support:0.2.0'
    implementation 'org.tensorflow:tensorflow-lite-metadata:0.1.0'
    implementation 'org.tensorflow:tensorflow-lite-gpu:2.5.0'

    implementation fileTree(dir: 'model')

    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```