1. Before making the app, have your tflite file ready.
2. Open Android studios and create a new project
    a. Select EMPTY VIEWS ACTIVITY (not Empty Activity)
    b. Keep the language as kotlin
    c. Use a good name
3. We won't waste time doing the UI, just copy the following code in **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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/main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:rotationX="0"
   android:visibility="visible"
   tools:context=".MainActivity"
   tools:visibility="visible">


   <ImageView
       android:id="@+id/myImagePreview"
       android:layout_width="match_parent"
       android:layout_height="250.dp"
       android:background="#000"
       android:contentDescription="@string/original_image" />

   <LinearLayout
       android:id="@+id/myTwoButtons"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_below="@+id/myImagePreview"
       android:layout_centerHorizontal="true"
       android:layout_centerVertical="true"
       android:orientation="horizontal">


       <Button
           android:id="@+id/captureBtn"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@string/capture_image" />

       <Space
           android:layout_width="30.dp"
           android:layout_height="wrap_content" />

       <Button
           android:id="@+id/improveBtn"
           android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="@string/improved_image" />

   </LinearLayout>

   <ImageView
       android:id="@+id/myOutputImage"
       android:layout_width="match_parent"
       android:layout_height="250.dp"
       android:layout_below="@+id/myTwoButtons"
       android:background="#000"
       android:contentDescription="@string/improved_image" />

   <TextView
       android:id="@+id/idTVMsg2"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_below="@+id/myOutputImage"
       android:gravity="center"
       android:padding="10dp"
       android:textAlignment="center"
       android:textColor="@color/red"
       android:textSize="20sp"
       android:textStyle="bold" />

</RelativeLayout>
```

4. Add the following 4 lines to **strings.xml**

```
<string name="capture_image">Capture Image</string>
<string name="enhance_images">Enhance Images</string>
<string name="improved_image">Improved image</string>
<string name="original_image">original image</string>
<string name="time_taken_to_run_the_model">Time taken to run the
model:</string>
```

5. In **MainActivity.kt**, add the following lines before OnCreate function:

```
private lateinit var captureBtn: Button
private lateinit var improveBtn: Button
private lateinit var myImagePreview: ImageView
private lateinit var myOutputImage: ImageView
private lateinit var bitmap: Bitmap
private lateinit var msgTV: TextView
private lateinit var msgTV2: TextView

var width: Int = 0
var height: Int = 0
lateinit var bitmap_lr: Bitmap
lateinit var bitmap_sr: Bitmap
```

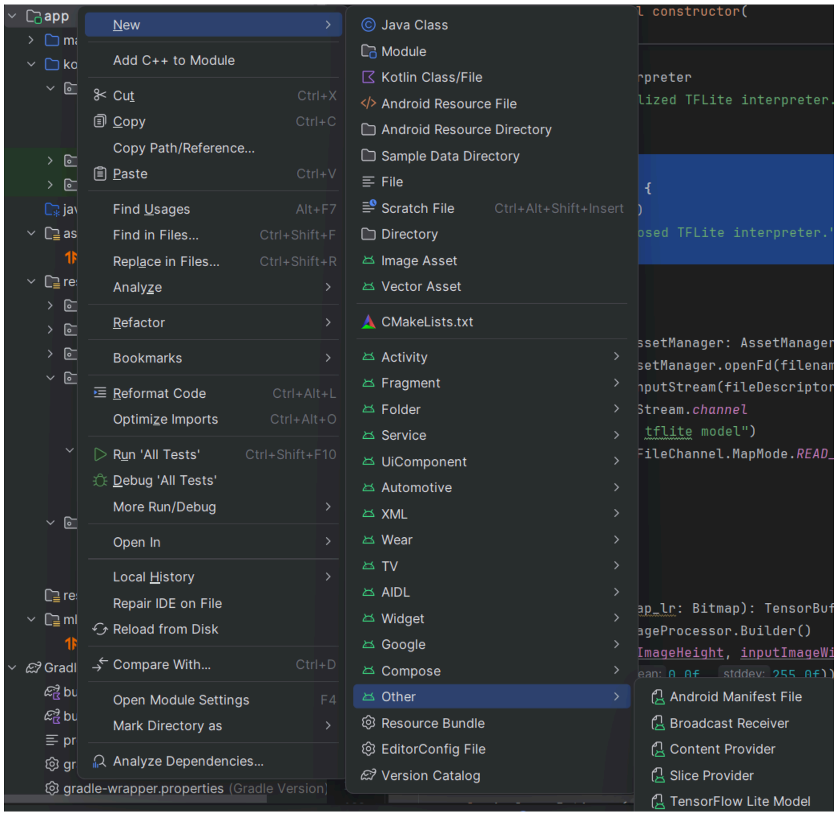
```
val TAG = "debugging"
val useGpu = false


lateinit var myContext: Context
```

6. Remove the ViewCompat and then Initialize the UI elements in the OnCreate function making it look like this:

```
@RequiresApi(Build.VERSION_CODES.R)
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   enableEdgeToEdge()
   setContentView(R.layout.activity_main)

   captureBtn = findViewById(R.id.captureBtn)
   improveBtn = findViewById(R.id.improveBtn)
   myImagePreview = findViewById(R.id.myImagePreview)
   myOutputImage = findViewById(R.id.myOutputImage)

   myContext = this

//more code will come here
}
```

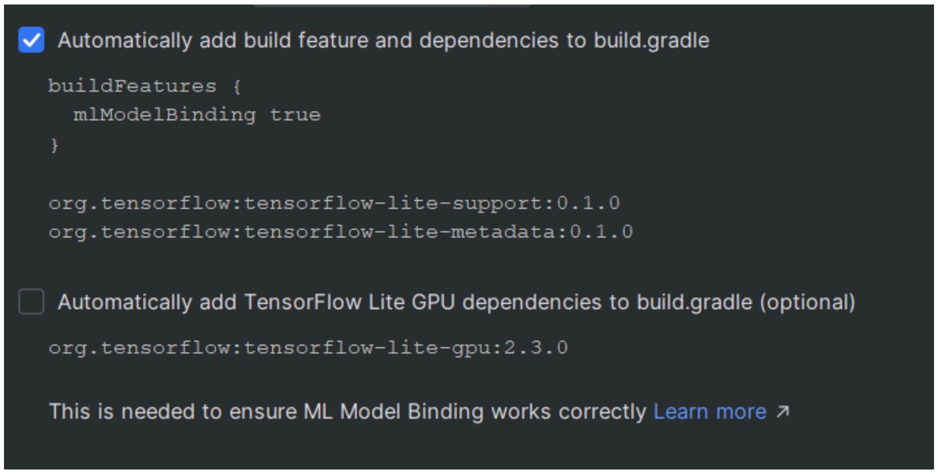
7. Let’s import the model now.

a. Right click on “app” and import tflite model



b. Select TensorFlowLite Model and pick your model.

c. DO NOT IMPORT THE TENSORFLOW LITE GPU DEPENDENCY, THEY HAVE DEPENDENCY ISSUES AND YOUR CODE WILL NOT WORK.!!!!



[✓] Automatically add build feature and dependencies to build.gradle

```
buildFeatures {
  mlModelBinding true
}

org.tensorflow:tensorflow-lite-support:0.1.0
org.tensorflow:tensorflow-lite-metadata:0.1.0
```

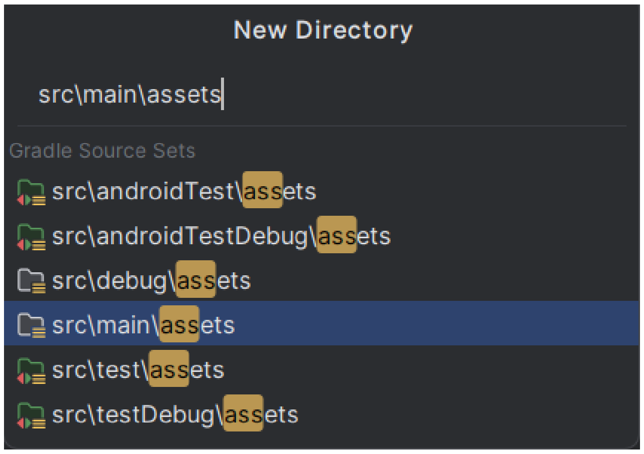
[ ] Automatically add TensorFlow Lite GPU dependencies to build.gradle (optional)

```
org.tensorflow:tensorflow-lite-gpu:2.3.0
```

This is needed to ensure ML Model Binding works correctly Learn more ↗

d.

e. Create a new directory in main called assets



New Directory

src\main\assets

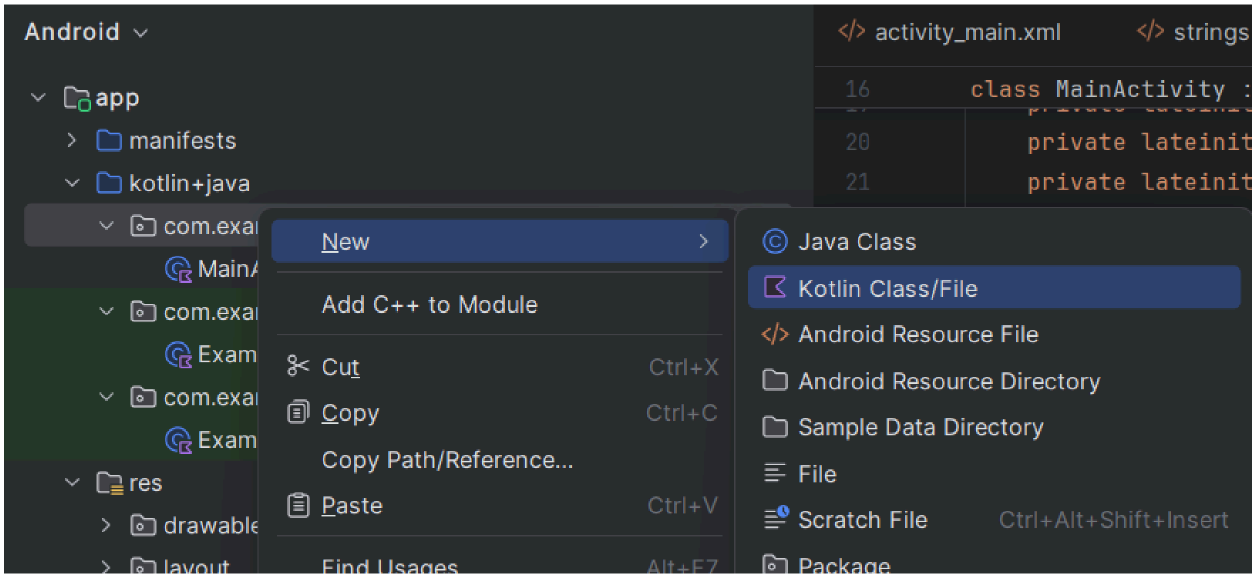
Gradle Source Sets

src\androidTest\assets
src\androidTestDebug\assets
src\debug\assets
src\main\assets
src\test\assets
src\testDebug\assets

f. Open the assets folder in Explorer (file explorer / file manager of your laptop)

g. Copy and paste the tflite model from ml directory to assets directory.

8. Before we go any further, let’s create some functions and classes which will be very useful.
9. Right click on the folder which contains your **MainActivity.kt**, click on new and make a kotlin class - I’m using the name: “**MyTFliteInterpreter**”.



```
class MainActivity :
        private lateinit
        private lateinit
```

10. Define the class:

```
class TFliteInterpreter internal constructor(
   private val assetManager: AssetManager,
   private val useGpu: Boolean,
   private val context: Context
) {


}
```

11. Initialize some variables and model name:

```
var interpreter: Interpreter? = null
```

```
private var options: Interpreter.Options? = null
private val executorService: ExecutorService = Executors.newCachedThreadPool()
private var inputImageWidth: Int = 0
private var inputImageHeight: Int = 0
private var modelInputSize: Int = 0
private lateinit var outputShape: IntArray


private val MODEL_NAME = "repnet.tflite" // name of your model's file

init {
   init()
}

companion object {
   private const val TAG = "debugging"
   private const val FLOAT_TYPE_SIZE = 4
}
```

12. For interpreter use: `import org.tensorflow.lite.Interpreter`
    And add the tensorflow dependency required.
13. Now let's write the initialization code for the class:

```
@SuppressLint("DefaultLocale")
@Throws(IOException::class)
private fun init() {
}
```

   a. Inside the init initialize the following variables:

```
@SuppressLint("DefaultLocale")
@Throws(IOException::class)
private fun init() {

options = Interpreter.Options()
val model = loadModelFile(context.assets, MODEL_NAME)
val interpreter = Interpreter(model, options!!)

val inputShape = interpreter.getInputTensor(0).shape()
outputShape = interpreter.getOutputTensor(0).shape()

inputImageWidth = inputShape[3]  // Adjusting for (batch, channels, height,
width)
inputImageHeight = inputShape[2]
modelInputSize = FLOAT_TYPE_SIZE * inputShape[1] * inputImageWidth *
inputImageHeight

this.interpreter = interpreter
//interpreter initialized
}
```

b. Function to close it:

```
fun close() {
   executorService.execute {
       interpreter?.close()
       Log.d(TAG, "Closed TFLite interpreter.")
   }
}
```

14. Let’s write a function to load the tflite model file into the 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
   //Log.d(TAG, "Loaded tflite model")
   return fileChannel.map(FileChannel.MapMode.READ_ONLY,
fileDescriptor.startOffset, fileDescriptor.declaredLength)
}
```

15. Define a simple run function which will run inference using the interpreter

```
fun run(a: Any?, b: Any?) {
   interpreter!!.run(a, b)
}
```

16. Now the confusing part, tensorflow interpreter expects inputs and outputs in very specific format and pytorch has inputs and outputs in NCHW format. We need to do the appropriate conversions and ensure that it works. The easiest to understand this conversion is to do it pixel by pixel.

```
fun prepareInputTensor(bitmap_lr: Bitmap): TensorBuffer {

       //allocate memory to create the ByteBuffer
       val buffer =
ByteBuffer.allocateDirect(modelInputSize).order(ByteOrder.nativeOrder())

       //make an array named "pixels" which will contain the rgb values of the
image
       val pixels = IntArray(inputImageWidth * inputImageHeight)
       bitmap_lr.getPixels(pixels, 0, bitmap_lr.width, 0, 0, bitmap_lr.width,
bitmap_lr.height)


       //bitmap is stored as ARGB => ((shift right by 16) & 0xFF) gives R
       //bitmap is stored as ARGB => ((shift right by 8) & 0xFF) gives G
       //bitmap is stored as ARGB => ((shift right by 0) & 0xFF) gives B
       //let's load the values into buffer one by one
```

```
        //since my model comes form pytorch, it expects input of form (1, 3,
720, 1280) hence my buffer is loaded like
        // R0 R1 R2 R3 ... G0 G1 G2... B0 B1 B2 ...
        //If my model were from tensorflow and needed input of form (1, 720,
1280, 3), my buffer would have to be loaded in the form:
        //R0 G0 B0 R1 G1 B1 R2 G2 B2 ....

        for (pixelValue in pixels) {
            buffer.putFloat((pixelValue shr 16 and 0xFF) / 255.0f) // loading
all pixels from Red channel
        }

        for (pixelValue in pixels) {
            buffer.putFloat((pixelValue shr 8 and 0xFF) / 255.0f)  // loading
all pixels from Green channel
        }

        for (pixelValue in pixels) {
            buffer.putFloat((pixelValue and 0xFF) / 255.0f)        // loading all
pixels from Blue channel
        }

        buffer.rewind() // resets the pointer of buffer to the beginning so that
it can be used again correctly.

        return TensorBuffer.createFixedSize(intArrayOf(1, 3, inputImageHeight,
inputImageWidth), DataType.FLOAT32).apply {
            loadBuffer(buffer) // bytebuffer to TensorBuffer (a fast memory type
which lies outside the scope of java)
        }
    }
```

17. Create an empty output tensorbuffer of the right shape:

```
fun prepareOutputTensor(): TensorBuffer {
    return TensorBuffer.createFixedSize(intArrayOf(outputShape[0],
outputShape[1], outputShape[2], outputShape[3]), DataType.FLOAT32)
}
```

18. Now let’s write the function which will convert the output Tensor Buffer to a bitmap image to store in our phone’s gallery.

```
fun tensorToImage(myBuffer: TensorBuffer): Bitmap {
    val buffer = myBuffer.buffer    //TensorBuffer to ByterBuffer
    buffer.rewind()     //start at the start

    val height = outputShape[2]
    val width = outputShape[3]

    // Prepare an array to store pixel values
    val pixels = IntArray(width * height)

    // Allocate space for channels
    val rChannel = FloatArray(width * height)
    val gChannel = FloatArray(width * height)
    val bChannel = FloatArray(width * height)

    // Read pixel values from ByteBuffer (NCHW format)
    for (i in 0 until width * height) {
        rChannel[i] = buffer.float // Read Red channel first
    }
    for (i in 0 until width * height) {
        gChannel[i] = buffer.float // Read Green channel next
    }
    for (i in 0 until width * height) {
        bChannel[i] = buffer.float // Read Blue channel last
    }

    // Convert to ARGB format (pixel by pixel)
    for (i in 0 until width * height) {
        val r = (rChannel[i] * 255.0f).toInt().coerceIn(0, 255)
        val g = (gChannel[i] * 255.0f).toInt().coerceIn(0, 255)
        val b = (bChannel[i] * 255.0f).toInt().coerceIn(0, 255)
        val a = 255 // Fully opaque

        // Store pixel in ARGB format
        pixels[i] = (a shl 24) or (r shl 16) or (g shl 8) or b
    }

    // Create Bitmap from pixel array
    val bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)
    bitmap.setPixels(pixels, 0, width, 0, 0, width, height)

    //Log.d(TAG, String.format("output image size %d %d",bitmap.width,
bitmap.height ))

    return bitmap
}
```

19. That is it for this class, just save the file and let’s go back to MainActivity.kt
20. Outside the OnCreate write a function called **runSR()** whose only job is to initialize an object of MyTFliteInterpreter class and run inference and measure the time.

```
@SuppressLint("DefaultLocale")
@RequiresApi(Build.VERSION_CODES.R)
@Throws(IOException::class)
private fun runSR() {

    val srModel: TFliteInterpreter = TFliteInterpreter(assets, useGpu,
myContext)
```

```
   // Prepare image by TensorImage
   val inputTensor: TensorBuffer = srModel.prepareInputTensor(bitmap_lr)
   val outputTensor: TensorBuffer = srModel.prepareOutputTensor()

   // Run the interpreter
   val startTime = System.currentTimeMillis()
   srModel.run(inputTensor.buffer, outputTensor.buffer)

   val time = System.currentTimeMillis() - startTime
   //Log.d(TAG, String.format("Spent time: %dms", time))

   // Show the result
   bitmap_sr = srModel.tensorToImage(outputTensor)
   myOutputImage.setImageBitmap(bitmap_sr)

   srModel.close()
   saveImage(bitmap_sr)

   msgTV2 = findViewById(R.id.idTVMsg2)

   msgTV2.text = String.format("time taken = %dms", time)
}
```

21. Write a function to save image to gallery:

```
@RequiresApi(Build.VERSION_CODES.R)
fun saveImage(bitmap: Bitmap) {
   val storageManager = getSystemService(STORAGE_SERVICE) as StorageManager
   val storageVolume = storageManager.storageVolumes[0] // internal Storage
   val fileImage = File(storageVolume.directory!!.path + "/Download/" +
System.currentTimeMillis() + ".jpeg")

   val byteArrayOutputStream = ByteArrayOutputStream()
   bitmap.compress(Bitmap.CompressFormat.JPEG, 100, byteArrayOutputStream)
   val bytesArray = byteArrayOutputStream.toByteArray()
   try {
       val fileOutputStream = FileOutputStream(fileImage)
       fileOutputStream.write(bytesArray)
       fileOutputStream.close()
       Log.d(TAG, "Image saved successfully")
   } catch (e: Exception) {
       throw RuntimeException(e)
   }
}
```

22. Now inside OnCreate:

   a. Write the OnClickListener for the buttons which will import and save the images

```
// button click listener
improveBtn.setOnClickListener { v: View? ->
```

```
    // Start Resolution when low resolution image is set
    Log.d(TAG, "Resolution Start!")
    try {
        runSR()
        Log.d(TAG, "success")
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

```
captureBtn.setOnClickListener{
    var intent: Intent = Intent()            //This is what helps to open the
gallery.
    intent.setAction(Intent.ACTION_GET_CONTENT)
    intent.setType("image/*")
    startActivityForResult(intent, 100)
}
```

23. Outside OnCreate lets define what happens when we click the buttons:

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?)
{
    super.onActivityResult(requestCode, resultCode, data)

    if(requestCode == 100){
        val uri = data?.data         //uri = uniform resource identifier
        if(uri != null) {
            bitmap = MediaStore.Images.Media.getBitmap(this.contentResolver,
uri)
            var x = 0
            var y = 0
            bitmap_lr = Bitmap.createBitmap(bitmap, x, y, 1280, 720)
            myImagePreview.setImageBitmap(bitmap_lr)
            Log.d("debugging", "Bitmap successfully imported / captured")
        } else {
            Toast.makeText(this, "Failed to get image!",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

24. In the **build.gradle.kts (:app)**:
    a. Change the compile and target SDK to 35
    b. Replace your current dependencies with this:

```
dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)
    implementation(libs.litert)
    implementation(libs.tensorflow.lite.support)
    implementation(libs.tensorflow.lite.metadata)
    implementation("org.tensorflow:tensorflow-lite:+")
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
}
```

25. In the **libs.verisons.toml** file replace the whole thing with the following code:

```
[versions]
agp = "8.5.0"
kotlin = "1.9.0"
coreKtx = "1.15.0"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
appcompat = "1.7.0"
material = "1.12.0"
activity = "1.10.0"
constraintlayout = "2.2.0"
litert = "1.1.2"
tensorflowLiteSupport = "0.1.0"
tensorflowLiteMetadata = "0.1.0"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref =
"coreKtx" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref =
"junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name =
"espresso-core", version.ref = "espressoCore" }
androidx-appcompat = { group = "androidx.appcompat", name = "appcompat",
version.ref = "appcompat" }
material = { group = "com.google.android.material", name = "material",
version.ref = "material" }
androidx-activity = { group = "androidx.activity", name = "activity",
version.ref = "activity" }
androidx-constraintlayout = { group = "androidx.constraintlayout", name =
"constraintlayout", version.ref = "constraintlayout" }
litert = { group = "com.google.ai.edge.litert", name = "litert", version.ref =
"litert" }
tensorflow-lite-support = { group = "org.tensorflow", name =
"tensorflow-lite-support", version.ref = "tensorflowLiteSupport" }
tensorflow-lite-metadata = { group = "org.tensorflow", name =
"tensorflow-lite-metadata", version.ref = "tensorflowLiteMetadata" }
```

```
[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
jetbrains-kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref =
"kotlin" }
```

26. Sync the gradle files.
27. Add red colour to **colors.xml**: `<color name = "red">#FFFF0000</color>`
28. Run the app.