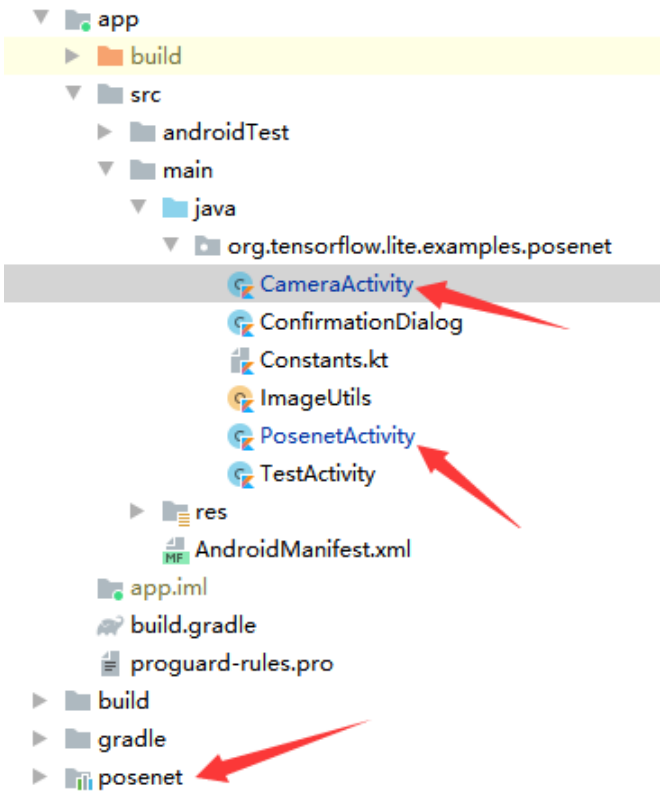
姿势估计

姿势估计可以估计图像中是否存在位置并输出人物的关键点(五官,躯干等)的位置

tensorflow官方提供的姿势估计demo中提供了一个posenet库,实际使用中如果没有特殊需求,可以直接使用这个库或者在此基础上稍加修改.接下来,将分别介绍这个posenet库在demo中的使用和实现

在demo中使用Posenet

项目文件结构



除了posent库,我们关注的其它代码都在app/src/main/java/org.tensorflow.lite.examples.posenet目录下,其中作为项目入口的是CameraActivity,但它很快打开了PosenetActivity,而我们的关键代码也主要在PosenetActivity中

```
class CameraActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.tfe_pn_activity_camera)
    savedInstanceState ?: supportFragmentManager.beginTransaction()
      .replace(R.id.container, PosenetActivity())
      .commit()
  }
}
```

Posenet类

在PosenetActivity中,我们主要做的事就是不断通过摄像头获取一张现实世界中的照片,并将照片输入模型,得到一个记录了照片中人物的"关键点"的位置的对象Person,然后根据Person中记录的位置在canvas画布上绘制人物的姿态.

为了向模型输入图形并获得返回值,我们声明了一个Posenet对象,通过调用Posenet对象的estimateSinglePose方法,我们就可以得到一个Person对象作为返回值

```
private fun processImage(bitmap: Bitmap) {
    // Crop bitmap.
    val croppedBitmap = cropBitmap(bitmap)

    // Created scaled version of bitmap for model input.
    val scaledBitmap = Bitmap.createScaledBitmap(croppedBitmap, MODEL_WIDTH,
MODEL_HEIGHT, true)

    // Perform inference.
    val person = posenet.estimateSinglePose(scaledBitmap)//获得Person对象
    val canvas: Canvas = surfaceHolder!!.lockCanvas()

    draw(canvas, person, scaledBitmap)//通过自定义函数draw,传入canvas,person和图像
scaledBitmap,绘制画面
  }
```

Person类

一个Person对象以一定的结构记录了某张图片上人物的位置,通过观察Posenet中的定义,我们可以总结出Person对象的结构 (这里我使用类似js中对象的形式表示了,这样看起来比较简洁和清楚(〃∀〃)):

```
{
  score: 0.32371445304906,//整个结果的置信度
  keyPoints: [
    {//这是一个KeyPoint对象
      position: {//这是一个Position对象
        y: 76.291801452637,
        x: 253.36747741699
      },
      bodyPart: NOSE,//这是一个BodyPart枚举对象
      score: 0.99539834260941//这一个KeyPoint的置信度
    },
    ...
  ]
}
```

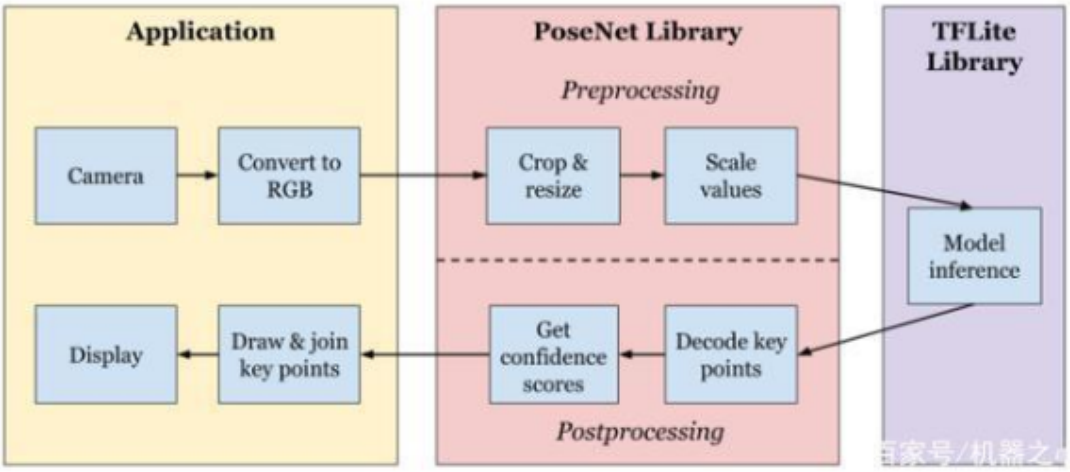
其中bodyPart是一个枚举对象,它用来形容这个关键点表示一个人身体的哪个部位

```
enum class BodyPart {
  NOSE,
  LEFT_EYE,
  RIGHT_EYE,
  LEFT_EAR,
  RIGHT_EAR,
  LEFT_SHOULDER,
  RIGHT_SHOULDER,
  LEFT_ELBOW,
  RIGHT_ELBOW,
  LEFT_WRIST,
  RIGHT_WRIST,
  LEFT_HIP,
  RIGHT_HIP,
  LEFT_KNEE,
  RIGHT_KNEE,
  LEFT_ANKLE,
  RIGHT_ANKLE
```

```
}
```

Posenet库的实现

Posenet库中最关键的一个类就是Posenet类,它为我们屏蔽了加载,调用posenet模型,处理图像和输出的结果的过程,我们通过estimateSinglePose方法输入图像并获得结果



Posenet类的getInterpreter()方法

```
private fun getInterpreter(): Interpreter {
    if (interpreter != null) {
        return interpreter!!
    }
    val options = Interpreter.Options()
    options.setNumThreads(NUM_LITE_THREADS)
    when (device) {
        Device.CPU -> { }
        Device.GPU -> {
            gpuDelegate = GpuDelegate()
            options.addDelegate(gpuDelegate)
        }
        Device.NNAPI -> options.setUseNNAPI(true)
    }
    interpreter = Interpreter(loadModelFile(filename, context), options)
    return interpreter!!
}
```

Posenet类有一个私有方法getInterpreter()用来得到一个Interpreter对象,Interpreter对象是Android app与tensorflow lite之间的桥梁,通过loadModelFile(filename, context)加载并返回一个MappedByteBuffer传给Interpreter

estimateSinglePose(bitmap: Bitmap): Person

就如前面所说,estimateSinglePose接收一个Bitmap对象并返回一个Person对象,

```
val inputArray = arrayOf(initInputArray(bitmap))
val outputMap = initOutputMap(getInterpreter())
getInterpreter().runForMultipleInputsOutputs(inputArray, outputMap)
```

我们通过上面的getInterpreter()方法获得posenet的Interpreter对象,并调用它的runForMultipleInputsOutputs喂入数据，得出结果存在outputMap中

outputMap是一个1 * x * y * z的浮点型数组:

```
/**
   * Initializes an outputMap of 1 * x * y * z FloatArrays for the model
processing to populate.
   */
  private fun initOutputMap(interpreter: Interpreter): HashMap<Int, Any> {
    val outputMap = HashMap<Int, Any>()

    // 1 * 9 * 9 * 17 contains heatmaps
    val heatmapsShape = interpreter.getOutputTensor(0).shape()
    outputMap[0] = Array(heatmapsShape[0]) {
      Array(heatmapsShape[1]) {
        Array(heatmapsShape[2]) { FloatArray(heatmapsShape[3]) }
      }
    }

    // 1 * 9 * 9 * 34 contains offsets
    val offsetsShape = interpreter.getOutputTensor(1).shape()
    outputMap[1] = Array(offsetsShape[0]) {
      Array(offsetsShape[1]) { Array(offsetsShape[2]) {
FloatArray(offsetsShape[3]) } }
    }

    // 1 * 9 * 9 * 32 contains forward displacements
    val displacementsFwdShape = interpreter.getOutputTensor(2).shape()
    outputMap[2] = Array(offsetsShape[0]) {
      Array(displacementsFwdShape[1]) {
        Array(displacementsFwdShape[2]) { FloatArray(displacementsFwdShape[3]) }
      }
    }

    // 1 * 9 * 9 * 32 contains backward displacements
    val displacementsBwdShape = interpreter.getOutputTensor(3).shape()
    outputMap[3] = Array(displacementsBwdShape[0]) {
      Array(displacementsBwdShape[1]) {
        Array(displacementsBwdShape[2]) { FloatArray(displacementsBwdShape[3]) }
      }
    }

    return outputMap
  }
```

最后,通过热力图heatmaps即outputMap[0]找到最有可能的关键点,并放入Person对象中,返回Person对象

```
    val heatmaps = outputMap[0] as Array<Array<Array<FloatArray>>>
    val offsets = outputMap[1] as Array<Array<Array<FloatArray>>>

    val height = heatmaps[0].size
    val width = heatmaps[0][0].size
    val numKeypoints = heatmaps[0][0][0].size
```

```
    // Finds the (row, col) locations of where the keypoints are most likely to
be.
    val keypointPositions = Array(numKeypoints) { Pair(0, 0) }
    for (keypoint in 0 until numKeypoints) {
      var maxVal = heatmaps[0][0][0][keypoint]
      var maxRow = 0
      var maxCol = 0
      for (row in 0 until height) {
        for (col in 0 until width) {
          if (heatmaps[0][row][col][keypoint] > maxVal) {
            maxVal = heatmaps[0][row][col][keypoint]
            maxRow = row
            maxCol = col
          }
        }
      }
      keypointPositions[keypoint] = Pair(maxRow, maxCol)
    }

    // Calculating the x and y coordinates of the keypoints with offset
adjustment.
    val xCoords = IntArray(numKeypoints)
    val yCoords = IntArray(numKeypoints)
    val confidenceScores = FloatArray(numKeypoints)
    keypointPositions.forEachIndexed { idx, position ->
      val positionY = keypointPositions[idx].first
      val positionX = keypointPositions[idx].second
      yCoords[idx] = (
        position.first / (height - 1).toFloat() * bitmap.height +
          offsets[0][positionY][positionX][idx]
        ).toInt()
      xCoords[idx] = (
        position.second / (width - 1).toFloat() * bitmap.width +
          offsets[0][positionY]
          [positionX][idx + numKeypoints]
        ).toInt()
      confidenceScores[idx] = sigmoid(heatmaps[0][positionY][positionX][idx])
    }

    val person = Person()
    val keypointList = Array(numKeypoints) { KeyPoint() }
    var totalScore = 0.0f
    enumValues<BodyPart>().forEachIndexed { idx, it ->
      keypointList[idx].bodyPart = it
      keypointList[idx].position.x = xCoords[idx]
      keypointList[idx].position.y = yCoords[idx]
      keypointList[idx].score = confidenceScores[idx]
      totalScore += confidenceScores[idx]
    }

    person.keyPoints = keypointList.toList()
    person.score = totalScore / numKeypoints

    return person
```

测试截图