移动互联网技术及应用

大作业报告

题目： ScAnimal —“野径”

野生动物识别系统的设计与实现

姓名	班级	学号
简旭锋	**2020211313**	**2020212895**

2023.5

目录

1. 相关技术

1.1 Google ML Kit Vision API

本项目中，使用 Google ML Kit Vision API 实现对于输入图像的预处理。

Google ML Kit Vision API 是 Google 提供的一种机器学习工具包，用于开发人工智能（AI）和机器学习（ML）功能的移动应用程序。它旨在帮助开发人员轻松地将视觉功能集成到他们的应用程序中，无论是在 Android 还是 iOS 平台上。

ML Kit Vision API 提供了一系列强大的视觉识别功能，包括图像分类、面部检测、条形码扫描、文本识别和图像标记等。开发人员可以使用这些功能来构建具有视觉智能的应用程序，例如自动识别物体、提取文字信息或者创建增强现实体验。

使用 ML Kit Vision API 时，开发人员只需几行代码就可以实现这些功能。API 内部使用了 Google 的先进机器学习模型和算法，这些模型经过训练可以识别各种类型的图像和对象。ML Kit Vision API 还支持离线功能，这意味着应用程序可以在没有网络连接的情况下运行。此外，它还提供了强大的实时功能，可以实时分析和处理相机捕获的视频流。

在本项目中，使用了 Google ML Kit Vision API 中的 ObjectDetection 功能，用于提取图像中的主体。

ML Kit Vision API 中的 Object Detection 功能可以识别和定位图像中的对象。它能够检测图像中的多个物体，并返回每个物体的位置和标签。以下是 Object Detection 的主要特点：

- **多类别检测：**Object Detection 能够同时检测多个物体，并将它们分类到预定义的类别中。这些类别可以包括常见的物体，如车辆、动物、家具等等。
- **物体定位：**除了识别物体的类别，Object Detection 还提供了物体的位置信息。它可以返回每个物体的边界框（bounding box），即一个矩形框，标识物体在图像中的位置和大小。
- **实时性能：**Object Detection 在移动设备上能够实时运行。它使用了轻量级的模型和优化的算法，以便在设备上高效执行。这使得开发人员能够构建具有实时对象检测功能的应用程序，例如增强现实应用或实时图像分析。
- **离线支持：**ML Kit Vision API 的 Object Detection 功能可以在离线环境下运行，不需要持续的网络连接。这使得应用程序能够在没有网络连接或者网络连接不稳定的情况下进行物体检测。
- **简单集成：**开发人员可以使用 ML Kit 提供的现成 API 来实现 Object Detection 功能，而无需编写复杂的机器学习代码。只需几行代码，就可以将 Object Detection 功能集成到应用程序中，为开发人员提供了一种方便且高效的方式，来识别和定位图像中的物体。

1.2 Baidu Animal API

本项目中，通过互联网请求，使用百度 API 平台中的动物识别 API，获取识别到的野生

动物的详细信息，包括品种、简介和百科链接。

百度 API 平台中的动物识别 API 是一项提供动物图像识别功能的人工智能服务。它基于百度强大的深度学习和计算机视觉技术，能够识别图像中的动物种类，并返回相应的结果。使用百度动物识别 API，开发人员可以通过发送包含动物图像的请求，获取关于该图像中动物的详细信息。以下是该 API 的主要特点：

- **动物分类：**动物识别 API 能够准确识别数百种不同的动物。它可以根据图像中的特征，将动物分类到相应的物种或品种中，提供准确的结果。
- **图像分析：**除了返回动物的类别，API 还可以提供有关图像的详细分析信息。例如，它可以指示图像中动物的位置、数量和其他相关特征，使开发人员可以获取更全面的图像分析结果。
- **高准确性：**百度动物识别 API 利用了深度学习技术，经过大量的训练和优化，具有较高的准确性。它能够识别图像中较为复杂的动物，具备较强的泛化能力。
- **简化集成：**百度 API 平台提供了简单易用的 API 接口，使开发人员可以轻松地将动物识别功能集成到应用程序中。只需通过 API 请求发送图像，即可获取相应的识别结果。
- **可定制性：**动物识别 API 还支持一些可选参数，例如返回结果中的置信度阈值和识别的顶级类别数等。开发人员可以根据自己的需求进行定制，获得最适合其应用的结果。

1.3 媒体访问与处理

在本项目中，使用了 Intent 和 MediaStore 以访问相机或图库获取图像。

当使用 Intent 和 MediaStore 打开相机和图库应用来获取图像时，我们利用了 Android 系统提供的相机和图库应用程序。这种方法具有以下特点和优势：

- **系统级应用程序：**通过使用 Intent 启动相机和图库应用，我们可以利用系统级应用程序的功能和性能。这些应用程序经过优化，并具备处理图像的能力，提供了相机拍摄和图像选择的标准界面。
- **设备兼容性：**由于使用了系统级应用程序，这种方法具有广泛的设备兼容性。几乎所有的 Android 设备都具备相机和图库应用，因此可以保证用户在大多数设备上能够进行图像获取操作。
- **用户友好性：**相机和图库应用具有熟悉的界面和交互方式，用户可以按照自己的习惯进行拍摄或选择图像。这样可以提供一致的用户体验，并减少开发人员需要编写和设计自定义界面的工作量。
- **图像处理支持：**使用 Intent 和 MediaStore 获取图像后，可以对返回的图像数据进行进一步的处理。例如，可以对图像进行裁剪、压缩、旋转或添加其他效果，以满足应用程序的特定需求。
- **扩展性：**通过使用 Intent 和 MediaStore，我们可以从相机和图库应用以外的第三方应用程序中获取图像。这使得应用程序可以利用其他应用程序的功能，例如社交媒体应用或云存储服务。

1.4 网络请求与响应处理

本项目中，使用了 Volley 库进行网络请求处理，以连接百度 API 获得识别结果。

通过 Volley 库进行网络请求是一种在 Android 应用中进行网络通信的常用方法。Volley 是由 Google 开发的网络请求库，旨在提供高效且简化的网络请求和响应处理机制。以下是对 Volley 库进行技术阐述的主要方面：

- **网络请求队列：**Volley 使用请求队列（RequestQueue）来管理网络请求。请求队列负责处理请求的调度和执行，可以同时处理多个请求，并自动处理请求的优先级、顺序和并发。
- **请求构建和发送：**通过创建不同类型的请求对象，如 StringRequest、JsonRequest 等，可以构建不同种类的网络请求。这些请求对象用于指定请求的 URL、方法（GET、POST 等）、参数、请求头、回调处理等信息，并通过请求队列发送到服务器。
- **响应处理：**Volley 库提供了简化的响应处理机制。可以设置请求的成功回调和错误回调，根据服务器返回的响应数据进行相应的处理，如解析 JSON、解析图片等。Volley 还支持自定义的响应解析器，以适应特定的数据格式和需求。
- **请求优化：**Volley 库通过多项优化措施来提高网络请求的性能和效率。例如，Volley 使用 HTTP 持久连接和连接池来减少网络连接的开销；它还支持请求的自动重试和请求的取消，以应对不稳定的网络环境。
- **线程管理：**Volley 库自动处理网络请求的线程管理，使开发人员无需手动处理线程相关的操作。Volley 使用线程池来管理网络请求的线程，并使用合适的线程进行请求和响应的处理，从而简化了线程管理的复杂性。

Volley 库提供了一种简单而强大的方式来进行网络请求和响应处理。通过使用 Volley，开发人员可以轻松地构建和发送网络请求，处理服务器的响应，并实现高效的图像加载功能。Volley 还通过优化措施和线程管理提供了良好的性能和用户体验，使网络通信在 Android 应用中变得更加简便和可靠。

1.5 视图绑定

本项目中，使用了 Android View Binding 技术以快速在代码中访问布局中的视图元素。

View Binding 是一种用于绑定视图（View）和布局文件的技术。它可以替代传统的 findViewById()方法，提供了一种类型安全、简洁且高效的方式来访问布局中的视图元素。View Binding 的主要原理是通过在编译时生成与布局文件对应的绑定类，该类包含了布局中的每个视图的引用。开发人员可以直接在代码中使用生成的绑定类来访问视图，而无需手动编写 findViewById()方法。以下是 View Binding 技术的一些关键点：

- **自动生成绑定类：**使用 View Binding，开发人员无需手动编写绑定代码。在编译时，Android 的编译器会自动生成与每个布局文件相关的绑定类。这些绑定类是根据布局文件的名称生成的，并与布局文件相关联。
- **类型安全访问：**通过 View Binding，开发人员可以使用绑定类直接访问布局文件中的视图元素，并且获得类型安全的引用。这意味着在编译时，如果布局文件中的视图 ID 发生更改

或不存在，将会在编译时报错，从而提前捕获错误。

- **优化性能：**由于 View Binding 是在编译时生成的，它具有很好的性能。与 findViewById()方法相比，View Binding 不需要在运行时进行视图查找操作，而是直接引用生成的绑定类中的视图。这可以提高应用程序的性能和响应速度。
- **支持包括布局文件和包含布局文件的模块：**View Binding 支持单个布局文件的绑定，也支持模块化开发中的包含布局文件的情况。它能够正确处理布局嵌套的情况，使开发人员可以轻松地访问包含在不同布局层次结构中的视图。
- **兼容性：**View Binding 是在 Android Gradle 插件版本 3.6.0 及更高版本中引入的，因此需要相应的构建工具版本支持。它与传统的 findViewById()方法兼容，可以在同一项目中同时使用它们。

View Binding 是一种在 Android 开发中提供类型安全、高效访问布局文件视图的技术。通过自动生成绑定类，它减少了手动编写繁琐的 findViewById()代码的工作量，并提供了更好的性能和开发效率。View Binding 使得访问布局视图变得更加简单、直观和可靠，有助于提高代码质量和可维护性。

2. 系统功能需求

2.1 系统功能

- **图像获取：**提供相机和相册的访问功能，使用户能够选择图像进行识别。
- **图像分割和物体识别：**进行图像分割和物体识别，将图像中的物体检测出来，并进行识别。
- **动物识别：**调用 Baidu 野生动物识别 API，将物体识别的结果传递给 API，获取详细的动物种类信息。
- **标注和展示：**在图像中以原点标注检测到的物体，并在用户点击时展示出物体的图像和识别到的动物种类。
- **百科信息：**在对话框中显示动物种类的名称和对应的百科页面，提供与动物相关的详细信息和知识。

2.2 用户需求分析

- **便捷性：**用户能够方便地使用相机或相册获取图像，并快速获得物体识别和动物种类识别的结果。
- **准确性：**系统能够准确地进行图像分割、物体识别和动物识别，提供准确的动物种类信息。
- **交互性：**用户能够通过点击图像上的标注点和动物种类，快速查看和浏览动物的详细信息。
- **可靠性：**系统能够稳定地进行图像处理和网络请求，并能够处理错误情况，如误识别或搜索失败。
- **用户界面友好性：**应用具有简洁直观的用户界面，使其易于理解和操作。

2.3 市场需求分析

- **教育和学习：**这样的应用对于对动物感兴趣的用户或学生来说，可以提供有趣的学习体验，并扩展他们的动物知识。
- **旅游和户外活动：**在户外活动中，用户可能会遇到一些野生动物，这样的应用可以帮助他们识别这些动物，并获取相关信息，提高他们的观察和了解能力。
- **自然保护和生态意识：**应用可以增强用户对野生动物的保护意识，让用户更加了解和关心野生动物的种类和生态环境。
- **娱乐和消遣：**用户可以将这样的应用用于娱乐和消遣，例如在郊游或户外活动中

3. 系统设计与实现

3.1 总体设计

3.1.1 应用架构

本项目使用 MVC（Model-View-Controller）架构进行应用构建。

MVC 模式将应用分为三个主要组件，通过定义清晰的职责划分，实现了模块之间的解耦和重用。

- Model（模型）：负责应用的数据和业务逻辑。它处理数据的读取、存储和处理，以及与数据相关的操作。
- View（视图）：负责应用的用户界面。它展示数据给用户，并接收用户的输入。
- Controller（控制器）：负责应用的逻辑控制。它接收用户的输入，对模型进行操作并更新视图。

具体的，在该野生动物识别系统中，MVC 应用架构可以描述为：

模型（Model）：

- **动物识别模型：**负责实现动物识别的算法和逻辑。它接收来自控制器的图像输入，使用图像处理和机器学习算法进行动物识别，并返回识别结果。
- **百科信息模型：**负责与百科数据库或 API 进行交互，获取与识别动物相关的准确的百科信息。它接收来自控制器的动物识别结果，根据识别结果查询相应的百科信息，并返回给控制器。

视图（View）：

- **动物识别界面：**展示用户界面，提供图像选择或拍摄的功能，并显示识别结果。它接收来自控制器的动物识别结果，并将结果展示给用户。
- **百科信息展示界面：**展示动物的详细百科信息，包括名称、描述、图片等。它接收来自控制器的百科信息，并将信息展示给用户。

控制器（Controller）：

- **动物识别控制器：**负责处理动物识别界面的用户输入和逻辑。它接收用户选择或拍摄的图像，并将图像传递给动物识别模型进行识别。一旦模型返回识别结果，控制器将结果传递给动物识别界面进行展示。
- **百科信息控制器：**负责处理百科信息展示界面的用户交互和逻辑。它接收来自动物识别控制器的识别结果，并调用百科信息模型获取相关的百科信息。一旦模型返回百科信息，控制器将信息传递给百科信息展示界面进行展示。

通过该 MVC 架构，应用程序按照模型、视图和控制器的角色进行分离，使得各个组件可以独立演化和测试。模型负责核心的业务逻辑和数据处理，视图负责用户界面的展示，控制器负责协调用户界面和模型之间的交互。这种分层结构有助于提高代码的可维护性、可测试性和可扩展性。

3.1.2 功能划分

- **图像选择：**通过打开相机或相册选取图像。
- **图像预处理：**对选择的图像进行分割以及对图像中物体的进行提取。
- **动物信息识别：**连接百度动物识别 API 获取准确的物种信息以及对应百科链接。
- **动物位置标记：**将识别后的动物位置标记在所选择的图像上，点击以查看详细数据。
- **动物数据展示：**呈现识别到的物种信息以及百科页面。

3.1.3 界面设计

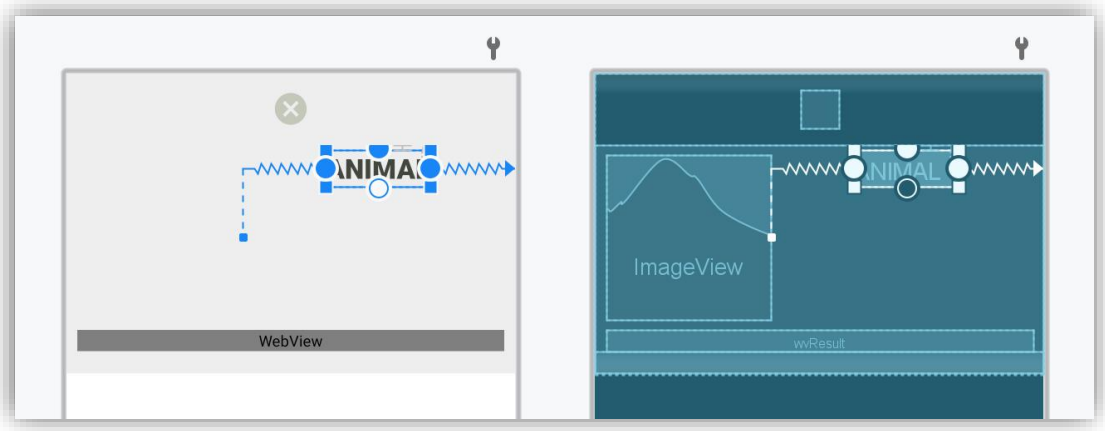
主页面设计

系统主页面通过 ConstraintLayout 进行构建。指定中心拍照按钮的位置后，其余组件均根据该按钮进行布局，并实现自动的居中和基线对齐。

选择图片后，图片会以合适的方式（长适配或宽适配）显示在图像组件中心。如果识别成功，会在识别到的物体中心显示圆点。点击圆点即可显示野生动物的详情页。

详情页设计

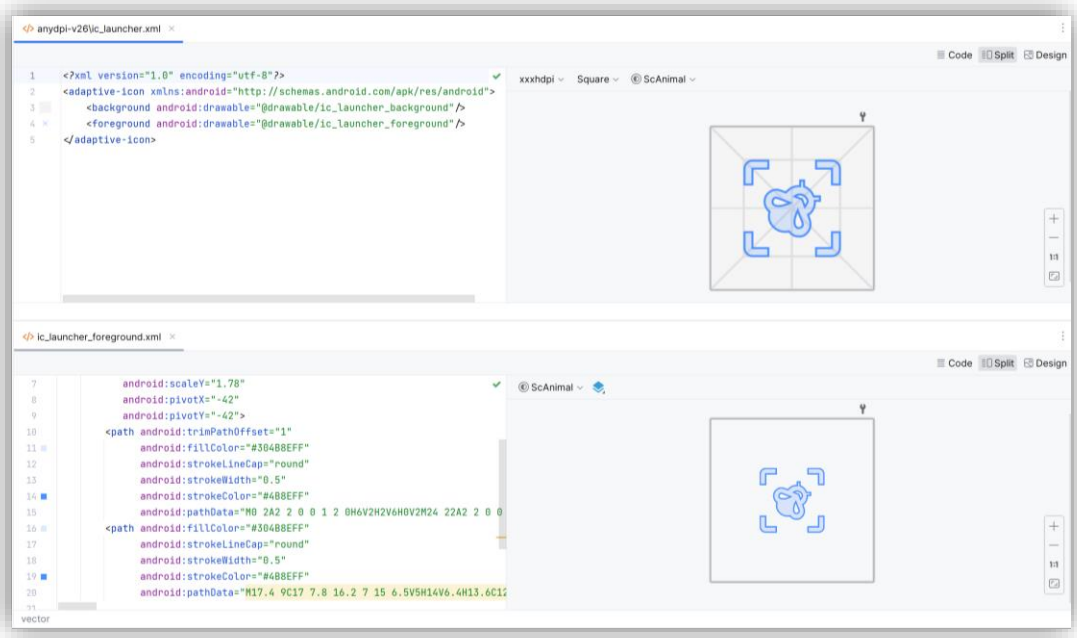
详情页通过 LinearLayout 构建。顺序布局按钮、简介栏以及网页。在简介栏中，通过 ConstraintLayout 完成文字的居中对齐。



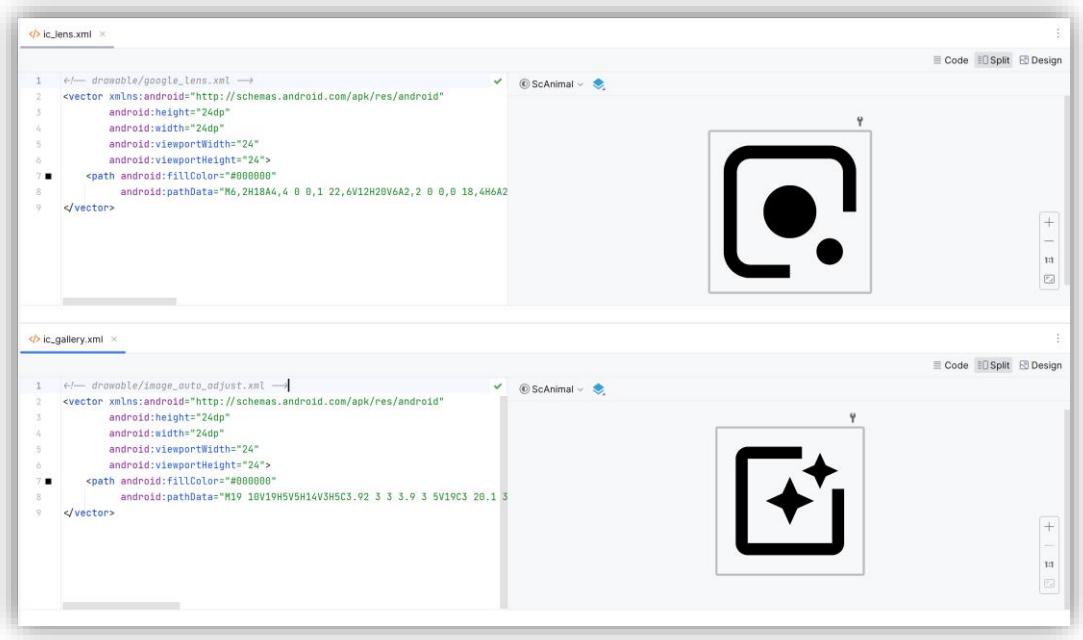
实际加载后，详情页的高度会设置为屏幕高度的 7/8，点击上方关闭按钮即可取消显示。在识别的所有结果中，点击其他结果即可切换到响应的百科详情页。

图标设计

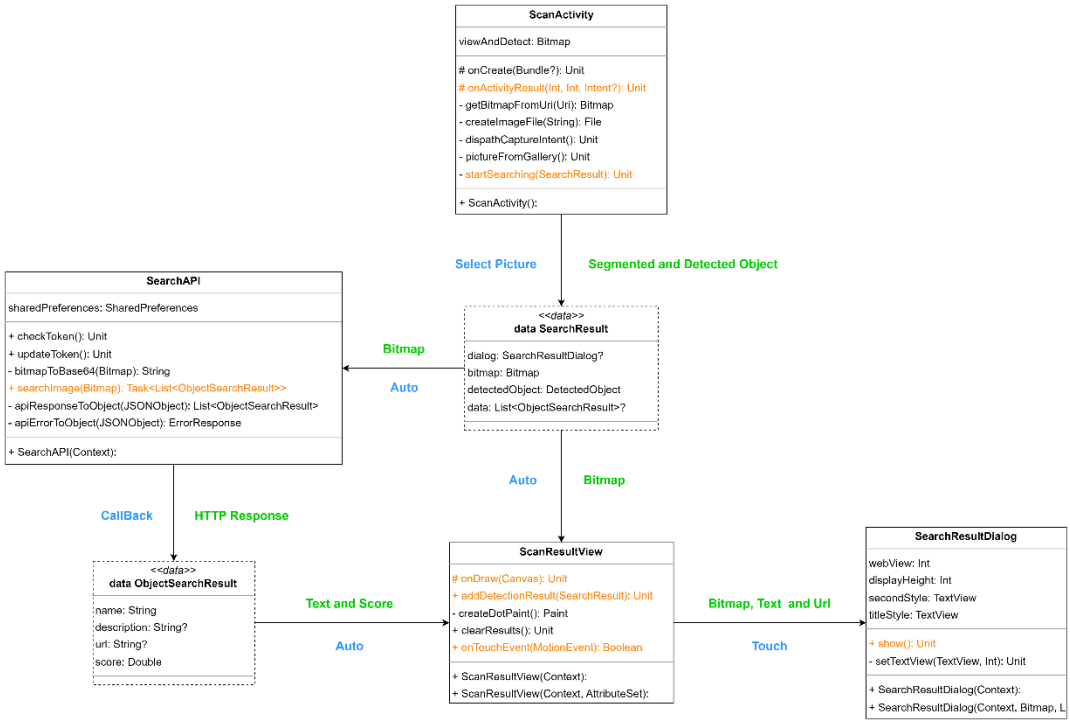
所有按钮图标以及应用图标均遵循 Google Material Design 设计，通过 xml 完成矢量图绘制。应用图标如下：



部分按钮图标如下：



3.1.4 模块关系与数据流



** 蓝色为控制流，绿色为数据流*

3.1.5 技术选型

本次项目使用 Kotlin 语言进行开发。Kotlin 是一种现代的静态类型编程语言，在 2017 年 5 月 17 日的 Google I/O 开发者大会上，Google 宣布将 Kotlin 作为一级官方支持的编程语言，与 Java 并列成为开发 Android 应用的首选语言。

特性

- **与 Java 的互操作性：**Kotlin 与 Java 高度兼容，可以直接在现有的 Java 项目中使用，无缝集成现有的 Java 库和框架。这意味着你可以逐步将 Java 代码迁移到 Kotlin，而无需从头开始重写整个项目。
- **简洁和表达力强：**Kotlin 采用了更简洁的语法，可以显著减少代码的数量。它引入了诸多现代化的语言特性，如空安全（null safety）、扩展函数（extension functions）、数据类（data classes）和 Lambda 表达式等，使得代码更加易读、易写和易维护。
- **空安全和类型推断：**Kotlin 提供了一种可靠的类型系统，可以在编译时捕获许多空指针异常，从而减少运行时错误。它引入了可空类型和非空类型的概念，以明确表达变量是否可以为 null。此外，Kotlin 还具有类型推断的能力，可以根据上下文自动推断变量的类型，减少了冗余的类型声明。
- **函数式编程支持：**Kotlin 在语言级别支持函数式编程，可以使用高阶函数、Lambda 表达式和集合操作等功能。这使得代码更加简洁、灵活，并且可以采用函数式编程的思维方式解决问题。
- **协程支持：**Kotlin 引入了协程（Coroutines）的概念，提供了一种轻量级的并发编程解决方案。协程可以简化异步操作的编写和管理，使得代码更加清晰和易于维护。

优势

相对于 Java，Kotlin 具有以下优势：

- **更少的样板代码：**Kotlin 通过简化语法和提供现代化的语言特性，减少了大量的样板代码，使得开发更加高效。
- **空安全：**Kotlin 的空安全类型系统可以避免空指针异常，提供更可靠的代码。
- **函数式编程支持：**Kotlin 内置了对函数式编程的支持，使得处理集合和异步操作等更加简洁和优雅。
- **可拓展性：**Kotlin 的语言设计具有可拓展性，可以轻松扩展现有的类和函数，而无需使用继承或工具类。
- **更好的工具支持：**Kotlin 在 Android Studio 等开发工具上具有良好的支持，包括智能代码补全、即时错误检查和自动重构等功能。

3.2 系统组成

系统由以下模块组成（模块关系与数据流详见 3.1.4）：

- **图像选择模块**：基于 Android Intent 和 Media Store 实现打开相机或相册选取图像。
- **图像预处理模块**：基于 Google ML Kit Vision API 实现图像分割以及对图像中物体的提取。
- **信息展示模块**：标记识别的动物位置，以及展示动物的物种和百科页面。
- **网络请求模块**：基于 Volley 完成 HTTP 请求，连接百度 API 获取准确的物种信息。

3.3 模块设计与实现

3.3.1 图像获取模块

模块功能

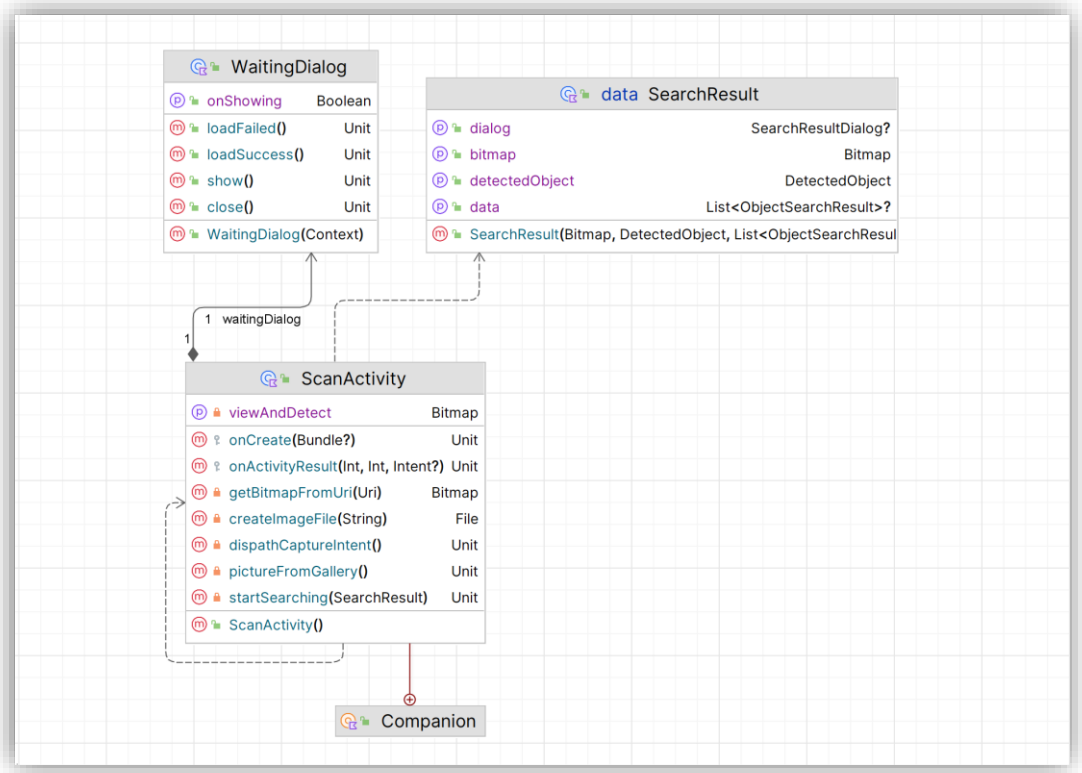
图像获取模块基于 Android Intent 和 Media Store 实现，基于此，可以完成打开相机拍摄图像或从相册选取图像的功能。

图像获取模块是该系统的入口活动，通过该活动完成后续所有过程和模块功能的调用和展示。基于该入口模块，系统实现了基于百度野生动物识别 API 的动物识别和搜索系统，通过拍照或选择照片进行动物识别，并使用 API 搜索相关的动物信息。系统具有用户界面交互、异步处理和错误处理等功能，以提供更好的用户体验。

模块接口

该模块的上层活动是 ScanActivity，作为系统的主活动，继承了 AppCompatActivity。

模块结构



其中，WaitingDialog 作为等待提示信息的展示组件，SearchResult 存储了动物识别的数据信息，其具体内容将在信息展示模块中说明。

核心函数

```
private fun dispathCaptureIntent(){
    Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent ->
        takePictureIntent.resolveActivity(packageManager)?.also {
            val photoFile: File? = try {
                createImageFile("TEMP_IMAGE")
            } catch (ex: IOException) {
                null
            }
            photoFile?.also {
                cameraPhotoUri = FileProvider.getUriForFile(this,
                    "com.labx.scanimal.fileprovider",it
                )
                takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, cameraPhotoUri)
                startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
            }
        } ?: run {
            Toast.makeText(this, "No Camera Found!", Toast.LENGTH_LONG).show()
        }
    }
}
```

具体来说，该模块的主要功能流程如下：

1. 首先，通过 **Intent(MediaStore.ACTION_IMAGE_CAPTURE)** 创建一个意图，该意图指定了要启动的是相机应用。
2. 通过 **takePictureIntent.resolveActivity(packageManager)** 检查设备上是否有相机应用可以处理该意图。
3. 如果有相机应用可用，进入代码块执行以下步骤：
   - 调用 **createImageFile("TEMP_IMAGE")** 创建一个临时图片文件，并将其赋值给 **photoFile** 变量。如果创建文件时发生异常，则 **photoFile** 的值将为 **null**。
   - 检查 **photoFile** 是否为非空值，如果是，则继续执行以下步骤：
     - 使用 FileProvider 为临时图片文件创建一个内容 URI，并将其赋值给 **cameraPhotoUri** 变量。FileProvider 是一种用于共享文件内容的特殊机制，以确保应用间的安全性。
     - 将内容 URI 添加为额外的输出，以便相机应用将拍摄的照片保存到指定的文件路径中。
     - 调用 **startActivityForResult (takePictureIntent, REQUEST_IMAGE_CAPTURE)** 启动相机应用，并希望在拍摄完成后返回结果。其中 **REQUEST_IMAGE_CAPTURE** 是一个整数请求码，用于在结果返回时识别该请求。
4. 如果没有相机应用可用，进入 **run** 代码块，显示一个 Toast 消息，提示用户找不到相机应用。

3.3.2 图像预处理模块

模块功能

经过图像获取模块获取到输入图像后，图像预处理模块会基于 Google ML Kit Vision API 完成对输入图像的分割和物体检测，并提取所有检测到的物体的边框信息以及大致分类。

该模块会对传入的图像进行操作，通过创建 ObjectDetection 实例调用 Google ML Kit Vision API 进行图像分割和物体识别，并对识别到的物体进行粗分类过滤，将非动物的物体剔除，随后根据各个物体的检测结果通过 createBimap 进行剪切，得到分离的独立物体图像，最后调用 startSearching 函数，通过其他 API 进行进一步生物识别。

模块接口

```
private fun setViewAndDetect(bitmap: Bitmap?)
```

在主活动 MainActivity 中选择输入图像后，系统会启动子线程启动该模块运行。

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(resultCode == RESULT_OK){
        when (requestCode) {
            REQUEST_IMAGE_CAPTURE -> cameraPhotoUri?.let {
                this.setViewAndDetect(getBitmapFromUri(it)
            }
            REQUEST_IMAGE_GALLERY -> data?.data?.let{
                this.setViewAndDetect(getBitmapFromUri(it))
            }
        }
    }
}
```

核心函数

```
Thread {
    val image = InputImage.fromBitmap(bitmap, 0)
    val options = ObjectDetectorOptions.Builder()
        .setDetectorMode(ObjectDetectorOptions.SINGLE_IMAGE_MODE)
        .enableMultipleObjects().enableClassification().build()
    val objectDetector = ObjectDetection.getClient(options)
    objectDetector.process(image)
        .addOnSuccessListener { results ->
            Thread{
                val filterResults = results.filter { result -> result.labels.size == 0 }
                detectResultNum = filterResults.size
                if(detectResultNum >= 6) {
                    runOnUiThread{waitingDialog.setFailedText("物体太多啦").loadFailed()}
                }else{
                    if(filterResults.size == 0) {
                        runOnUiThread{waitingDialog.setFailedText("没有发现小动物").loadFailed()}
                    }
                    filterResults.forEach{
                        var cropBitmap = Bitmap.createBitmap(
                            bitmap,it.boundingBox.left,it.boundingBox.top,
                            it.boundingBox.width(),it.boundingBox.height())
                        startSearching(SearchResult(cropBitmap,it, emptyList(),null))
                    }
                }
            }.start()
        }
    ...
```

具体来说，该模块的功能流程可以描述如下：

1. 首先，将 **detectResultNum** 和 **searchedResultNum** 的值都设置为 0，用于追踪检测和搜索结果的数量。
2. 调用 **viewBinding.ivPreview.clearResults()** 清除之前的检测结果，准备显示新的图片。
3. 检查传入的 **bitmap** 是否为非空值，如果是，则执行以下步骤：
   - 将 **bitmap** 设置为 **viewBinding.ivPreview** 的图像显示。
   - 显示一个等待对话框，提示用户正在进行搜索操作。
   - 创建一个后台线程，在该线程中执行以下步骤：
     - 创建一个 **InputImage** 对象，使用传入的 **bitmap** 创建一个输入图像。
     - 创建一个 **ObjectDetectorOptions** 对象，配置目标检测器的选项，包括设置检测模式为单张图像模式、启用多目标检测和分类功能。
     - 创建一个 **ObjectDetector** 对象，使用配置的选项。
     - 调用 **objectDetector.process(image)** 开始处理输入图像，返回一个 **Task** 对象。

- 在成功的回调中，获取检测到的结果列表，然后在一个新的后台线程中执行以下步骤：
  - 过滤结果列表，只保留没有标签的结果。
  - 更新 **detectResultNum** 的值为过滤后的结果数量。
  - 如果 **detectResultNum** 大于等于 6，则表示检测到的物体太多，显示一个警告消息，并在 UI 线程中更新等待对话框的状态为“加载失败”。
  - 否则，如果过滤后的结果列表为空，表示没有检测到小动物，显示一个失败消息，并在 UI 线程中更新等待对话框的状态为“加载失败”。
  - 否则，遍历过滤后的结果列表，对每个结果执行以下步骤：
    - 根据结果的边界框信息从原始图像中裁剪出一个子图像（小动物图像）。
    - 调用 **startSearching** 函数开始对裁剪的小动物图像进行搜索。
- 在失败的回调中，打印错误信息，并在等待对话框正在显示时，在 UI 线程中更新等待对话框的状态为“加载失败”。

4. 如果传入的 **bitmap** 为空值，函数直接返回。

3.3.3 网络请求模块

模块功能

网络请求模块通过 Volley 库完成 HTTP 请求。主要使用 requestQueue 进行请求的异步发送和接收，通过 Gson 进行响应消息的解析。

基于该网络请求，该模块可以**通过百度野生动物识别 API 实现动物信息搜索，对图像进行动物识别并获取识别结果。**系统通过 SearchAPI 类封装了与 API 的通信逻辑，包括获取访问令牌、发送识别请求和解析响应结果。

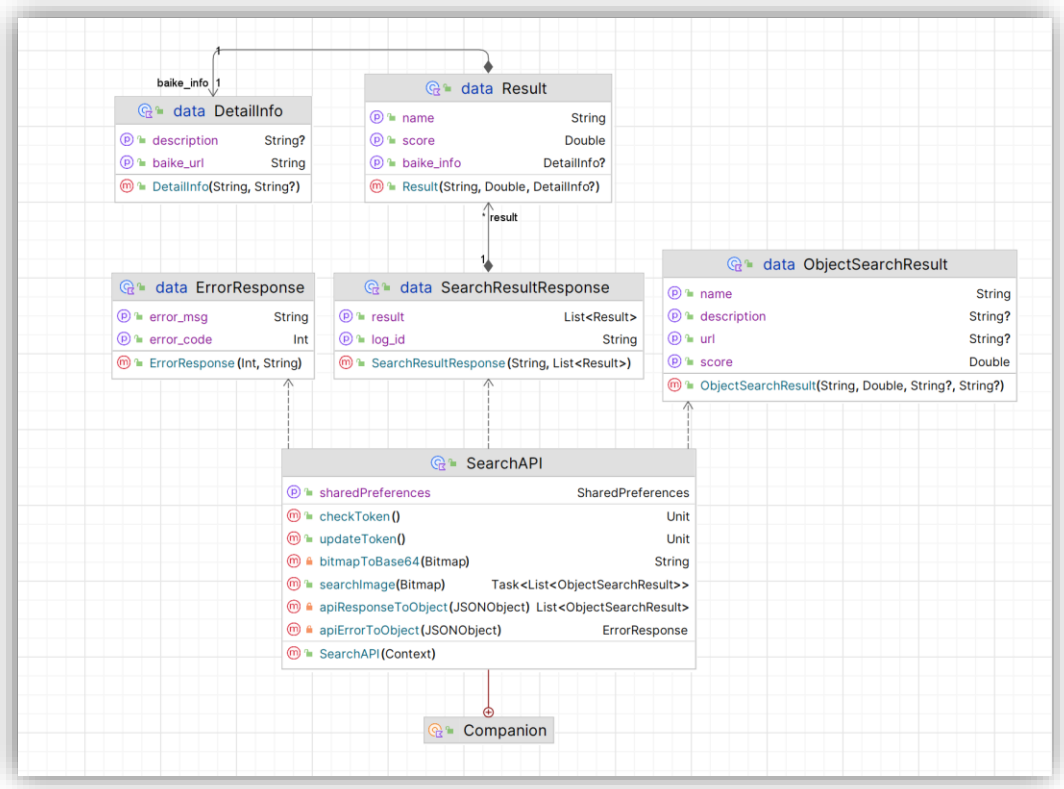
模块接口

```
fun searchImage(image: Bitmap): Task<List<ObjectSearchResult>>
```

在主活动 ScanActivity 中，通过图像预处理模块获得的检测结果会分别启动新的线程，针对分割出的物体图像进行详细的动物信息搜索。

```
private fun startSearching(searchResult: SearchResult){
    searchResult?.let {
        searchApi.searchImage(it.bitmap)
            .addOnSuccessListener {response ->  ...
```

模块结构



其中，SearchAPI 是该模块的主要调用接口，其余 data 类作为解析 HTTP 返回信息的基础结构，ObjectSearchResult 作为请求返回的识别信息的存储容器。

核心函数

```
fun searchImage(image: Bitmap): Task<List<ObjectSearchResult>> {
    val apiSource = TaskCompletionSource<List<ObjectSearchResult>>()
    val apiTask = apiSource.task
    var base64: String = bitmapToBase64(image)
    var base64Url = URLEncoder.encode(base64, "UTF-8")
    val token: String = sharedPreferences.getString("token", "")?: ""
    val req = "image=${base64Url}&top_num=3&baike_num=3"
    requestQueue.add(object :
        JsonObjectRequest(Method.POST,"$SEARCH_API_URL?access_token=${token}",null,{ response ->
            if(response.has("error_code")){
                val errorResponse = apiErrorToObject(response)
                apiSource.setException(Exception(errorResponse.error_msg))
            } else {
```

```
                val searchList = apiResponseToObject(response)
                apiSource.setResult(searchList)
            }
        },{ error ->apiSource.setException(error) }
    ) {
    override fun getBodyContentType() = "application/x-www-form-urlencoded"
    override fun getBody(): ByteArray {return req.toByteArray()}
}.apply {setShouldCache(false)})
return apiTask
}

private fun apiResponseToObject(response: JSONObject): List<ObjectSearchResult> {
    val objectSearchResults = mutableListOf<ObjectSearchResult>()
    val searchResult = Gson().fromJson(response.toString(), SearchResultResponse::class.java)

    searchResult.result.forEach {
        objectSearchResults.add(
            ObjectSearchResult(it.name, it.score,it.baike_info?.baike_url,it.baike_info?.description)
        )
    }
    return objectSearchResults
}

private fun apiErrorToObject(response: JSONObject): ErrorResponse {
    val errorResponse = Gson().fromJson(response.toString(), ErrorResponse::class.java)
    return errorResponse
}
```

该函数的主要功能流程大致为：

1. **searchImage** 函数的功能是搜索给定的图像，并返回一个 **Task<List<ObjectSearchResult>>** 对象。
   - 创建一个 **TaskCompletionSource<List<ObjectSearchResult>>** 对象 **apiSource**，用于处理搜索结果的异步任务。
   - 通过 **apiSource.task** 获取与 **apiSource** 关联的任务对象 **apiTask**。
   - 将输入的图像转换为 Base64 编码的字符串，并进行 URL 编码。
   - 从共享首选项中获取访问令牌 **token**。
   - 发送搜索请求，使用 **JsonObjectRequest** 发起一个 POST 请求，包含图像数据和其他参数。
   - 在请求成功的回调中，解析响应数据，并将解析后的搜索结果列表设置为 **apiSource** 的结果。
   - 在请求失败的回调中，将错误信息设置为 **apiSource** 的异常，并记录错误信息。
   - 返回 **apiTask** 对象，以便在调用函数的地方可以监听搜索结果。

2. **apiResponseToObject** 函数的功能是将搜索响应的 JSON 数据转换为 **List<ObjectSearchResult>** 对象。
   - 创建一个可变的 **productSearchResults** 列表，用于存储搜索结果。
   - 使用 Gson 将响应数据转换为自定义的 **SearchResultResponse** 对象。
   - 遍历 **SearchResultResponse** 对象的结果列表，将每个结果转换为 **ObjectSearchResult** 对象，并添加到 **productSearchResults** 列表中。
   - 返回 **productSearchResults** 列表。
3. **apiErrorToObject** 函数的功能是将错误响应的 JSON 数据转换为 **ErrorResponse** 对象。
   - 使用 Gson 将响应数据转换为自定义的 **ErrorResponse** 对象。
   - 返回 **ErrorResponse** 对象。

3.3.4 信息展示模块

模块功能

信息展示模块作为主要的交互界面，承载了系统的显示与交互功能。该系统中，信息展示模块主要分为两个组件：

ScanResultView：展示通过“图像选择模块”所选择的图片，并在识别结束后在图像中相应物体的位置显示标记。系统通过自定义的 ScanResultView 视图类，在图像上绘制可点击的圆点，用于显示物体的识别结果，并支持点击圆点展示详细信息。

ScanResultDialog：通过一个自定义的底部弹窗对话框，**展示图像识别结果的详细信息和相关百科网页。**用户在 ScanResultView 中点击对应物体的圆点后，系统会弹出底部弹窗，显示分割出的图像和识别到的物体种类。用户可以点击物体种类，在对话框中展示对应的百科网页。

模块接口

```
fun addDetectionResult(result: SearchResult)
```

在动物识别结束后，系统会将非动物数据过滤，并将识别为动物的检测结果通过该接口添加到结果展示模块中，后续该模块将会调用 onDraw 方法进行动物标记的绘制。

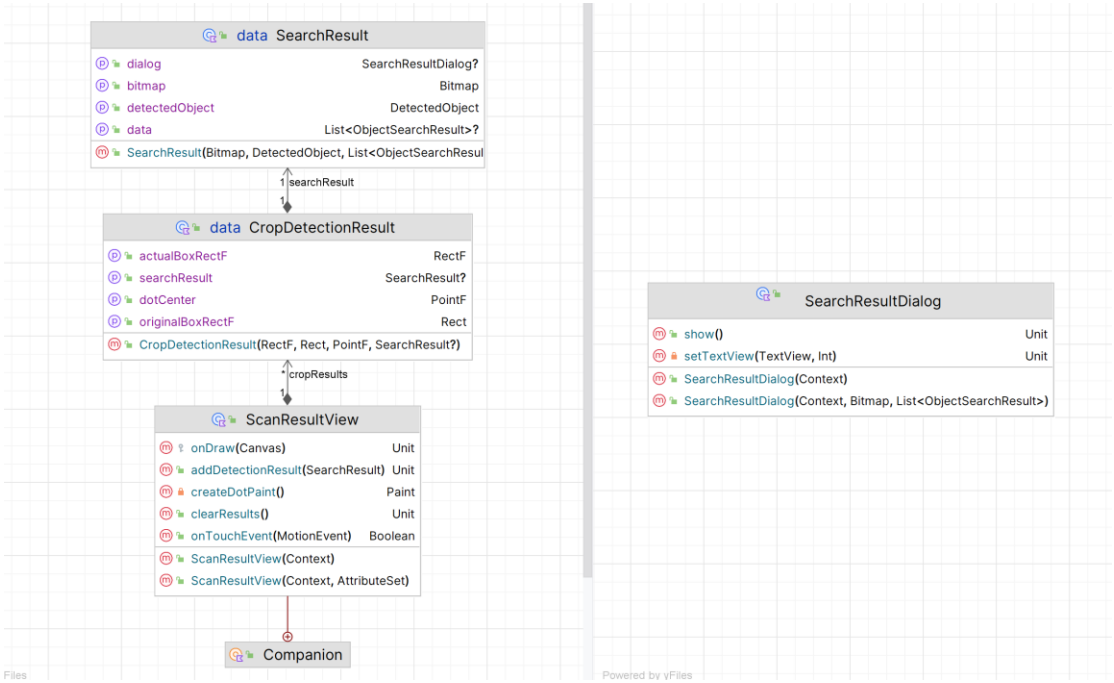
```
if((response.size > 0) && (response[0].name != "非动物")){
    it.data = response
    it.dialog = SearchResultDialog(this,it.bitmap,response)
    synchronized(this){
        viewBinding.ivPreview.addDetectionResult(it)
    }
    ...
```

```
override fun onTouchEvent(event: MotionEvent): Boolean
```

在圆点绘制后，通过点击圆点，系统会将动物识别的具体信息通过底部弹出的对话框进行展示，并同时设置对话框中的百科网页的内容。

```
fun show(){
    setTextView(viewbind.title,0)
    setWebView(0)
    bottom_sheet.show()
}
```

模块结构



核心函数

```
override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)
    cropResults.forEach { result ->
        canvas.drawCircle(result.dotCenter.x, result.dotCenter.y, CLICKABLE_RADIUS, dotPaint)
    }
}
```

```
override fun onTouchEvent(event: MotionEvent): Boolean {
    when (event.action) { MotionEvent.ACTION_DOWN -> {
            val touchX = event.x; val touchY = event.y
            val index = cropResults.indexOfFirst {
                    val dx = (touchX - it.dotCenter.x).toDouble().pow(2.0)
                    val dy = (touchY - it.dotCenter.y).toDouble().pow(2.0)
                    (dx + dy) < CLICKABLE_RADIUS.toDouble().pow(2.0)
                }
            if (index != -1) { cropResults[index]?.searchResult.let { it?.dialog?.show() } }
        }
    }
    return super.onTouchEvent(event)
}
```

具体的，该函数的功能和流程大致可以描述为：

1. **onDraw** 函数的功能是在自定义的 View 上绘制内容。
   - 首先，调用父类的 **super.onDraw(canvas)** 方法，确保绘制默认的视图内容。
   - 使用一个循环遍历 **cropResults** 列表，其中存储了物体识别结果对象。
   - 对于每个裁剪结果对象，使用 **canvas.drawCircle** 方法在指定位置绘制一个圆形，圆心的坐标由 **result.dotCenter.x** 和 **result.dotCenter.y** 提供，半径为 **CLICKABLE_RADIUS**，画笔样式为 **dotPaint**。
   - 绘制完成后，自定义的视图包含一系列圆点，每个圆点代表一个识别到的动物。
2. **onTouchEvent** 函数的功能是处理触摸事件。
   - 当触摸事件发生时，根据事件的类型（**event.action**）进行处理。
   - 如果触摸动作为 **MotionEvent.ACTION_DOWN**，即手指按下的动作：
     - 获取触摸点的坐标，分别保存为 **touchX** 和 **touchY**。
     - 使用 **indexOfFirst** 方法在 **cropResults** 列表中找到第一个满足条件的裁剪结果对象索引。
     - 如果找到了满足条件的裁剪结果对象（即索引不为 -1）：
       - 获取该裁剪结果对象的 **searchResult** 属性，并使用 **let** 函数执行操作。
       - 在 **searchResult** 对象中，调用 **dialog.show()** 方法显示对话框。
     - 如果未找到满足条件的裁剪结果对象，不执行任何操作。
   - 返回 **super.onTouchEvent(event)**，确保继续处理触摸事件的默认行为。

4. 系统可能的扩展

针对现有系统，可以考虑以下方式来进一步扩展该应用：

- **增加更多的物体识别功能：**除了野生动物，可以扩展应用的物体识别能力，识别更多的物体类别，例如植物、建筑、食物等，以满足用户对不同领域的需求。

- **增强动物识别准确性：**通过改进图像分割和物体识别算法，提升系统对动物的识别准确性，减少误识别的情况，提供更可靠的识别结果。

- **用户社区功能：**创建用户社区平台，让用户可以分享他们的拍摄图像和识别结果，与其他用户交流和讨论。这可以增加用户之间的互动性，同时也为其他用户提供更多有趣的图像和识别案例。

- **扩展多平台支持：**除了 Android 平台，将应用扩展到其他平台，如 iOS 和 Web，以便更多用户可以使用和享受该应用。

- **AR 增强现实功能：**结合增强现实技术，让用户可以通过手机摄像头实时观察到物体标注和动物识别结果，提供更沉浸式的体验。

- **跨平台集成和数据共享：**与其他相关应用或平台进行集成，例如与旅游应用、动物保护组织合作，共享数据和资源，提供更全面和综合的服务。

通过不断扩展和改进现有系统，可以提供更多功能和增强用户体验，使应用更具吸引力和实用性。根据用户反馈和市场需求，持续关注和改进系统，保持与时俱进。

5. 总结体会

移动互联网技术是指将互联网与移动通信技术相结合，实现在移动设备上进行信息传递、数据交互和服务提供的技术体系。**通过移动互联网技术，人们可以随时随地通过移动设备获取信息、进行在线交流、享受各种移动应用和服务。**

在完成《移动互联网技术及应用》课程的期末作业中，我设计并开发了名为 **ScAnimal——“野径”的野生动物识别应用。**通过这个过程，我对移动互联网技术有了更深入的理解和认识，并获得了一些重要的体会。

首先，我意识到移动互联网技术的普及和便捷性给人们的生活带来了巨大的变化。随着智能手机的普及和移动网络的发展，移动互联网已经成为人们生活中不可或缺的一部分。通过开发这款应用，我深刻体会到移动互联网技术的便捷性和普适性，用户可以随时随地通过手机获取信息、进行交互和分享。通过智能手机和移动网络，我们可以随时随地浏览新闻、社交媒体、在线购物、娱乐等。移动互联网技术使得信息和服务无处不在，极大地方便了人们的生活和工作。

其次，我对移动应用开发技术有了更深入的了解。在开发过程中，我学习了 Android 开发平台和相关技术，如图像处理、物体识别、API 调用等。这些技术使得我能够实现应用的各种功能，例如图像分割和识别、调用第三方 API 获取动物种类信息等。通过实际开发，我对移动应用的架构、界面设计、数据处理等方面有了更多的实践经验。

另外，移动互联网技术的发展离不开与第三方服务的集成。通过与各种第三方服务和 API 的集成，我们可以为移动应用增加更多的功能和服务，拓展应用的边界。在本次开发中，通过调用 Baidu 野生动物识别 API，我可以在应用中获取准确的动物种类信息，并展示详细的百科信息。这让我认识到在移动互联网应用中，与第三方服务的集成是非常重要的，可以拓展应用的功能和提供更全面的服务。

此外，移动互联网技术对于用户需求和体验的重视也非常明显。用户是移动应用的核心，他们的需求和体验直接影响应用的使用和推广。在移动互联网应用的开发过程中，我们需要深入了解用户的需求，设计用户友好的界面和交互方式，并不断优化应用的性能和用户体验。通过数据分析和个性化推荐，我们可以更好地满足用户的需求，提供更加个性化和精准的服务。

总的来说，通过参与《移动互联网技术及应用》课程的学习和期末作业的设计与开发，我对移动互联网技术有了更深入的理解和认识。通过学习和实践，我学会了使用 Android 开发平台和相关技术进行移动应用开发，了解了移动互联网技术的应用场景和重要性，也认识到用户需求和体验在移动应用中的关键地位，以及与第三方服务的集成和数据分析的重要性。移动互联网技术是一个日益发展和重要的技术领域，对于人们的生活和工作产生了深远的影响。移动互联网技术将继续发展壮大，为人们带来更多便利和创新的应用和服务。