

# 目录

一 序言 . . . . .	1
二 研究目的及意义 . . . . .	1
三 Tensorflow Lite . . . . .	2
四 设计方案 . . . . .	3
五 关键代码解释 . . . . .	3
六 总结 . . . . .	8
七 致谢 . . . . .	9



南昌航空大學

# 课程设计报告

题 目： 基于 TensorFlow Lite 开发的的图像识别程序

学 院： 测试与光电工程学院

专业名称： 生物医学工程

班级学号： 19084129 19084127

学生姓名： 李奕澄 骆小宇

指导教师： 尚璇

填表日期 2023 年 1 月 27 日

## **一、序言**

移动应用的开发在近年来发展迅速，几乎所有用户常用的功能都已经有相应的软件能够满足需求。但与此同时，用户对应用的要求也相应提高，而不仅仅满足于功能实现。移动应用除了在功能和性能上要有所突破外，在用户隐私保护和安全性上的要求也相应提高。如何以更加高效合理的方式开发，高性能且安全的应用以满足用户的需求成为了每个移动开发者的首要任务。与此同时，基于深度学习的人工智能在近年来快速发展，让许多技术领域有了重大突破。如何和人工智能进行有效结合，让移动开发有了新的尝试途径。传统的图像分类应用都需要将图像通过网络上传到云端服务器，再进行图像分类并将分类标签重新下发到移动端，移动端再将分类结果根据标签进行展示。这样的做法不仅依赖网络，低效并且存在安全隐患。为了克服上述问题，本程序设计和实现一个可以在移动端本地对用户的物品进行识别分类的 Android 应用软件。该软件在移动端通过 Tensorflow Lite 神经网络框架运行图像分类神经网络对用户的物品进行实时识别。

**关键词** Android, Tensor flow lite, 图像识别

## **二、研究目的及意义**

基于 TensorFlow Lite 的 Android 图像识别应用程序是一个热门研究项目，对计算机视觉和移动应用程序开发领域具有重大意义。在这个项目中，机器学习模型使用 TensorFlow（一种流行的开源机器学习框架）进行训练，然后转换为 TensorFlow Lite 格式以部署在移动设备上。由此产生的图像识别应用程序能够使用 Android 设备的摄像头实时识别物体。

其主要优势是，它可以在资源有限的移动设备上实现实时对象检测。这有许多潜在的应用，例如在增强现实、自动驾驶汽车和机器人领域。此外，该研究项目还展示了如何将机器学习模型集成到移动应用程序中，以便开发人员可以创建更智能、更具交互性和响应用户需求的应用程序。

该研究项目的另一个重要方面是迁移学习的使用，其预先训练的机器学习模型通过在小数据集上微调其参数来适应特定任务。这种方法减少了对大量训练数据和计算资源的需求，这在资源有限的移动和嵌入式系统中尤为重要。此外，迁移学习可以通过利用从预训练模型中学到的知识来提高机器学习模型的准确性和效率。

Tensorflow Lite 突出了将机器学习模型集成到移动应用程序中的潜力。移动设备已成为我们日常生活中越来越重要的一部分，对能够为用户提供有用服务和体验的智能和个性化移动应用程序的需求不断增长。通过将机器学习模型集成到移动应用程序中，开发人员可以创建更智能、更具交互性和响应用户需求的应用程序。

最后，围绕 Tensorflow Lite 开发对可以在移动和嵌入式设备上高效运行的机器学习模型具有重要意义。因为在这些设备的计算资源和内存有限。而 TensorFlow Lite 可以为在移动设备上运行机器学习模型提供轻量级和优化的运行环境，这可以显着降低所需的能耗和处理时间。

### 三、Tensorflow Lite

TensorFlow Lite 是流行的开源机器学习框架 TensorFlow 的轻量级版本，专为在移动和嵌入式设备上运行机器学习模型而设计。它提供了一组工具和库，用于在资源有限的设备上构建、优化和部署机器学习模型，例如智能手机、平板电脑、智能手表和物联网（IoT）设备。

与完整版 TensorFlow 不同，TensorFlow Lite 专为在功能强大的服务器和工作站上进行高性能计算而设计，TensorFlow Lite 针对在低功耗和有限处理能力的设备上运行进行了优化。TensorFlow Lite 可用于广泛的机器学习任务，包括图像识别、对象检测、自然语言处理和语音识别。它拥有许多功能和优势，使其成为在资源有限的设备上构建机器学习应用程序的有吸引力的选择。

相对于其他图像识别技术，TensorFlow Lite 具有以下优势：

1. 轻量级：TensorFlow Lite 是专门为移动和嵌入式设备设计的轻量级框架。它占用空间小，非常适合部署在智能手机和物联网设备等资源受限的设备上。
2. 快速高效：TensorFlow Lite 针对移动和嵌入式设备进行了优化，使其在处理速度和功耗方面快速高效。这允许在边缘设备上实时执行机器学习模型。
3. 易于使用：TensorFlow Lite 易于集成到现有的移动应用程序中，并支持广泛的平台，包括 Android, iOS 和 Linux。
4. 与 TensorFlow 兼容：TensorFlow Lite 建立在 TensorFlow 框架之上，可以轻松地将 TensorFlow 模型转换为 TensorFlow Lite 模型，以便在移动和嵌入式设备上部署。

而 TensorFlow Lite 在拥有以上有点情况同时，还有以下缺点：

1. 功能有限：与完整版的 TensorFlow 相比，TensorFlow Lite 的功能有限。它仅支持 TensorFlow 操作的子集，不支持分布式计算。
2. 模型大小受限：由于移动和嵌入式设备的约束，可以在 TensorFlow Lite 上部署的机器学习模型的大小受到限制。这可能会使在这些设备上部署更复杂的模型变得具有挑战性。
3. 可定制性较差：TensorFlow Lite 专为在移动和嵌入式设备上部署而设计，并针对特定用例进行了优化。与完整版的 TensorFlow 相比，这可能会使其可定制性降低。
4. 支持有限：TensorFlow Lite 是一个相对较新的框架，因此，与完整版的 TensorFlow 相比，它的文档和社区支持有限。
5. 无法使用部分硬件加速：例如 GPU、TPU、XLA 等。

总体而言，TensorFlow Lite 是一个轻量级高效的框架，专为在移动和嵌入式设备上部署而设计。虽然它在功能和模型尺寸方面有一些限制，但它非常适合移动和嵌入式领域的广泛应用。

## 四、设计方案

针对所需内容，本项目采用 TensorFlow Lite 作为图像识别的框架，使用 Android Studio 作为开发环境，以 Java 语言开发，使用 Android 手机作为测试设备。本项目的设计方案如下：

1. 收集和准备数据集：下一步是收集和准备数据集。这涉及收集与用例相关的大型图像数据集并相应地标记它们。数据集应该是多样化的，并代表应用程序将遇到的实际方案。
2. 训练机器学习模型：收集和准备数据集后，下一步是使用 TensorFlow 训练机器学习模型。这涉及创建一个深度学习模型架构，并使用 TensorFlow 的 API 在数据集上对其进行训练。
3. 将模型转换为 TensorFlow Lite：模型经过训练后，需要将其转换为 TensorFlow Lite 格式才能部署在移动设备上。这可以使用 TensorFlow 的转换工具完成。
4. 将模型集成到 Android 应用程序中：下一步是将 TensorFlow Lite 模型集成到 Android 应用程序中。这涉及将模型添加到应用程序的代码库，并为用户创建与模型交互的接口。

本次设计的关键点在于创建深度学习模型架构、Tensorflow 的 API 调用以及数据集的收集以及调用。而数据集则来源于网络上的公开数据集，如 Github 等。

## 五、关键代码解释

针对本项目的文件以及代码，我们在此对其进行了解释，以便读者能够更好地理解本项目的实现过程。

```
package org.tensorflow.lite.examples.imageclassification;
```

这个包使用了预训练的模型，可以识别 1000 种图像类别。这个包还包含了一个 Android 测试类，用于验证图像分类结果是否与预期一致。

在 androidTest 中写的是测试程序，用于验证模型的分类结果是否与预期一致。测试程序使用了 JUnit 框架，并使用了 Android 测试库。

以下代码用于导入 JUnit 框架中的一些静态方法。这些方法可以用来测试程序。使用静态导入可以省略类名，可以方便地访问类中的静态成员。

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
```

以下代码验证图像分类器进行的分类是否与预期的控制类别匹配。首先检查分类结果是否为空，然后比较控制数据和分类数据中的类别数。最后，遍历每个类别并比较标签。

```
assertNotNull(results.get(0));  
  
assertEquals(controlCategories.size(),  
            results.get(0).getCategories().size());  
  
for (int i = 0; i < results.size(); i++) {  
    assertEquals(  
        controlCategories.get(i).getLabel(),  
        results.get(0).getCategories().get(i).getLabel()  
    );  
}
```

以下代码为可简化加载预训练的图像分类模型并使用它来对图像进行分类的过程，其创建一个实例并传递一个侦听分类结果的对象。

```
ImageClassifierHelper helper = ImageClassifierHelper.create(  
    InstrumentationRegistry.getInstrumentation().getContext(),  
    new ImageClassifierHelper.ClassifierListener() {  
        @Override  
        public void onError(String error) {  
            // no-op  
        }  
  
        @Override  
        public void onResults(  
            List<Classifications> results,  
            long inferenceTime  
        ) {  
            // ...  
        }  
    });
```

以下代码使用了 main 文件夹中 ImageClassifierHelper 类，其封装了 TensorFlow Lite 图像分类任务的辅助类。其加载了 coffee.jpg 的图片，并将其传递给 classify 方法进行分类。classify 方法会调用 onResults 方法来处理分类结果和推理时间，并与控制数据进行比较。如果分类结果和控制数据不一致，测试就会失败。

```
helper.setThreshold(0.0f);  
helper.classify(loadImage("coffee.jpg"), 0);
```

此代码用于从 assets 文件夹加载图像并返回对象。该方法用于加载将由图像分类器分类的图像。

```

private Bitmap loadImage( String fileName ) {
    AssetManager assetManager = InstrumentationRegistry
        .getInstrumentation()
        .getContext()
        .getAssets();

    try {
        InputStream inputStream = assetManager.open(fileName);
        return BitmapFactory.decodeStream(inputStream);
    } catch ( IOException e ) {
        e.printStackTrace();
    }
    return null;
}

```

文件夹 main/assets 里包含的是模型文件和标签文件，这些文件是在训练模型时生成的。模型文件是一个二进制文件，包含了模型的权重和结构信息。标签文件是一个文本文件，包含了模型可以识别的 1000 种图像类别。

针对 CameraFragment 类，用于设置相机及其用例，并允许用户修改分类分数阈值、一次可分类的最大对象数、用于分类的线程数以及用于对象分类的底层硬件和模型等设置。

以下代码用于检查应用是否具有访问设备相机所需的权限。如果没有，它将导航到使用组件 PermissionsFragment Navigation

```

if ( ! PermissionsFragment . hasPermission( requireContext() ) ) {
    Navigation . findNavController( requireActivity() , R . id .
fragment_container )
        . navigate(
            CameraFragmentDirections .
actionCameraToPermissions()
        );
}

```

以下代码负责在销毁片段视图时执行清理操作。

```

@Override
public void onDestroyView() {
    super.onDestroyView();

    // Shut down our background executor
    cameraExecutor.shutdown();
    synchronized ( task ) {
        imageClassifierHelper . clearImageClassifier();
    }
}

```

以下代码为分析用例捕获的每个相机帧调用该方法。它将传递给以对图像中的对象进行分类。如果检测到对象并将其分类到阈值以上，则它们将显示在屏幕底部的回收器视图中。

```

private void classifyImage(@NonNull ImageProxy image) {
    // Copy out RGB bits to the shared bitmap buffer
    bitmapBuffer.copyPixelsFromBuffer(image.getPlanes()[0].getBuffer());

    int imageRotation = image.getImageInfo().getRotationDegrees();
    image.close();
    synchronized (task) {
        // Pass Bitmap and rotation to the image classifier helper
        for {
            // processing and classification
            imageClassifierHelper.classify(bitmapBuffer, imageRotation)
        ;
        }
    }
}

```

`ClassificationResultAdapter` 类是用于显示分类结果列表的自定义 `RecyclerView` 适配器的实现。适配器包含类别列表和适配器大小，并公开两个用于更新这些属性的公共方法。

该方法接收对象列表，并使用这些对象更新适配器的列表。它首先使用自定义实现按类别索引对给定列表进行排序。然后，它使用 `updateResults`、`Category`、`categories`、`Comparator`、`ArrayList`、`adapterSize` 变量，表示可以在回收器视图中显示的最大类别数。

然后，该方法循环遍历排序的类别，并将它们添加到 `adapterSize` 或大小 `sortedCategories` 列表，以较小者为准。最后，它调用 `notifyDataSetChanged()` 以通知回收器视图数据集已更改，需要重新绘制。

这 `adapterSize` 具有给定大小的变量。此方法用于更新回收器视图中可以显示的最大类别数。

```

@SuppressLint("NotifyDataSetChanged")
public void updateResults(List<Category> categories) {
    List<Category> sortedCategories = new ArrayList<>(categories);
    Collections.sort(sortedCategories, new Comparator<Category>() {
        @Override
        public int compare(Category category1, Category category2) {
            return category1.getIndex() - category2.getIndex();
        }
    });
    this.categories = new ArrayList<>(Collections.nCopies(
adapterSize, null));
    int min = Math.min(sortedCategories.size(), adapterSize);
    for (int i = 0; i < min; i++) {
        this.categories.set(i, sortedCategories.get(i));
    }
    notifyDataSetChanged();
}
public void updateAdapterSize(int size) {
    adapterSize = size;
}

```

`PermissionsFragment` 类用于请求应用访问设备相机所需的权限。并在必要时请求它该方法检查是否已向应用授予权限。如果已授予权限，它将调用 `navigateToCamera`。如果尚未授予权限，它将使用 `requestPermissionLauncher` 向用户请求权限。

```
@Override  
public void onStart() {  
    super.onStart();  
    if (ContextCompat.checkSelfPermission(getApplicationContext(),  
        Manifest.permission.CAMERA) == PackageManager.  
    PERMISSION_GRANTED) {  
        navigateToCamera();  
    } else {  
        requestPermissionLauncher.launch(Manifest.permission.CAMERA  
    );  
    }  
}
```

`ImageClassifierHelper` 类有助于使用 TensorFlow Lite 完成图像分类任务。它包括用于设置和运行图像分类器的方法，以及用于传回分类结果和错误消息的侦听器接口。在前面已有部分解释。

在 `setupImageClassifier` 方法中首先创建一个对象 `ImageClassifierOptions.Builder`，为图像分类器设置各种选项，例如分数阈值和最大结果数。

```
@Override  
ImageClassifier.ImageClassifierOptions.Builder optionsBuilder =  
ImageClassifier.ImageClassifierOptions.builder()  
    .setScoreThreshold(threshold)  
    .setMaxResults(maxResults);
```

接下来，创建 `BaseOptions.Builder` 来为委托设置各种选项，例如要使用的线程数。

```
BaseOptions.Builder baseOptionsBuilder =  
BaseOptions.builder().setNumThreads(numThreads);
```

根据 `currentDelegate` 的值，该方法选择要使用的委托（CPU、GPU 或 NNAPI），并在 `BaseOptions.Builder` 中设置适当的选项。

```
switch (currentDelegate) {  
    case DELEGATE_CPU: break;  
    case DELEGATE_GPU:  
        if (new CompatibilityList().isDelegateSupportedOnThisDevice  
()) {  
            baseOptionsBuilder.useGpu();  
        } else {imageClassifierListener.onError("GPU is not  
supported on "+ "this device");}  
        break;  
    case DELEGATE_NNAPI:  
        baseOptionsBuilder.useNnapi();  
}
```

根据 currentModel 的值，该方法选择要使用的模型（MobileNetV1 或 EfficientNetLite 的三个版本之一），并设置适当的模型文件名。

```
String modelName;
switch (currentModel) {
    case MODEL_MOBILENETV1:
        modelName = "mobilenetv1.tflite";
        break;
    case MODEL_EFFICIENTNETV0:
        modelName = "efficientnet-lite0.tflite";
        break;
    case MODEL_EFFICIENTNETV1:
        modelName = "efficientnet-lite1.tflite";
        break;
    case MODEL_EFFICIENTNETV2:
        modelName = "efficientnet-lite2.tflite";
        break;
    default:
        modelName = "mobilenetv1.tflite";
}
```

最后，ImageClassifier 尝试从所选模型和选项创建对象。如果成功，则 imageClassifier 的字段设置为新对象。如果不成功，imageClassifierListener 将向控制台报告错误消息，并将错误日志打印到控制台。

```
try {
    imageClassifier =
        ImageClassifier.createFromFileAndOptions(
            context,
            modelName,
            optionsBuilder.build());
} catch (IOException e) {
    imageClassifierListener.onError("Image classifier failed to "
        + "initialize. See error logs for details");
    Log.e(TAG, "TFLite failed to load model with error: "
        + e.getMessage());
}
```

## 六、总结

本文主要围绕 Tensorflow Lite 进行简单介绍以及分析，将其所拥有的优点和缺点进行了对比，并且对其发展前景进行预测。在此基础上，本文围绕 Tensorflow Lite 进行了简单的设计和实现，设计了一个可以在移动端本地对用户的物品进行识别分类的 Android 应用软件。并且对其进行简单的封装，使得其在移动端的使用更加方便。最后，本文对 Tensorflow Lite 的使用进行了简单的测试，测试结果表明，Tensorflow Lite 在移动端的使用效果良好，可以满足用户的需求。

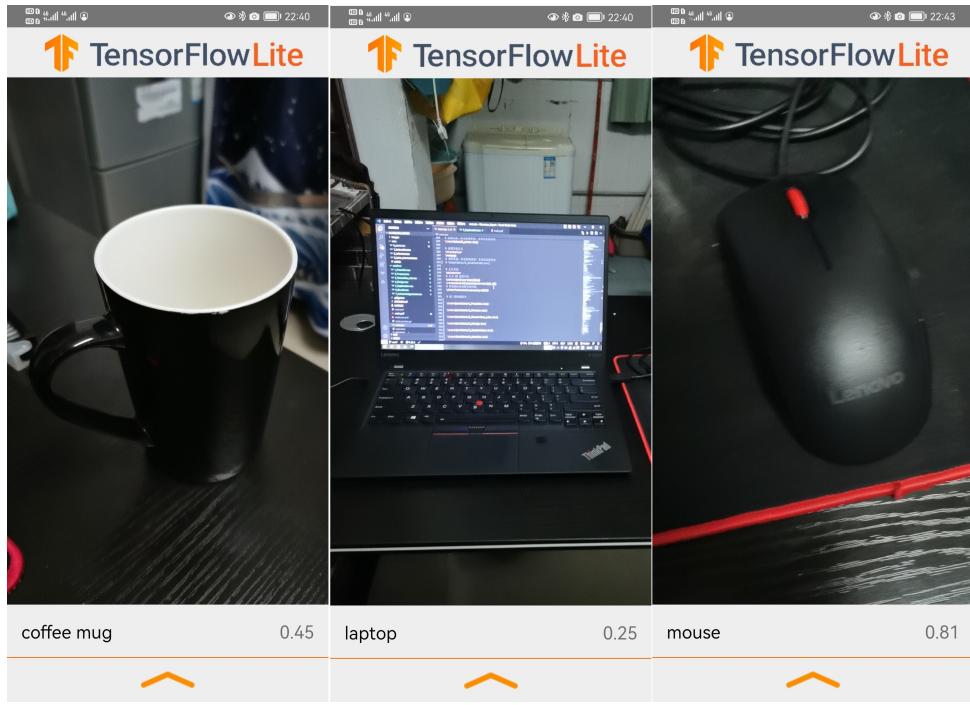


图 1 实际效果

针对本次设计仍存在缺点，如：

- 图像识别的准确率仍然需要提高。由于模型的训练图像集都是经过挑选的主体明显的图像，和实际识别的复杂场景有较大区别。所以模型要进一步提升实际图像识别状况下的准确率。
- 软件功能不成熟。虽然应用有实验性质，但是软件功能较少，设计比较粗糙，交互不够人性化。
- 程序设计结构可以优化。其直接影响程序运行速度。

通过本次课程设计学习了 Tensorflow Lite 的相关知识以及操作，对 Tensorflow Lite 的使用有了一定的了解，并且通过动手设计了一个简单程序，从中收获良多。

## 七、致谢

最终要感谢在整个课程设计过程中帮助过我所有人。首先，也是最主要感谢的是我的指导老师，尚璇老师。在整个过程中她给了我很大的帮助，在课设计划制定时，她首先确定了我的题目大方向，但是同时又帮我详细分析使我有了明确的目标，尚璇老师丰富的教学经验和严谨的治学之道，潜移默化的影响了我的学习生涯。在课程设计的过程中，尚璇老师的全力支持、严格要求和悉心指导，使我的课程设计得以完成。其次也非常感谢帮助过我的室友和同学，在我遇到问题时帮助我走出困境，这次做课设的经历也会使我终身受益，我感受到做课程设计，是真正的自己学习的过程和研究的过程，希望这次的经历能让我在以后学习中激励我继续进步。