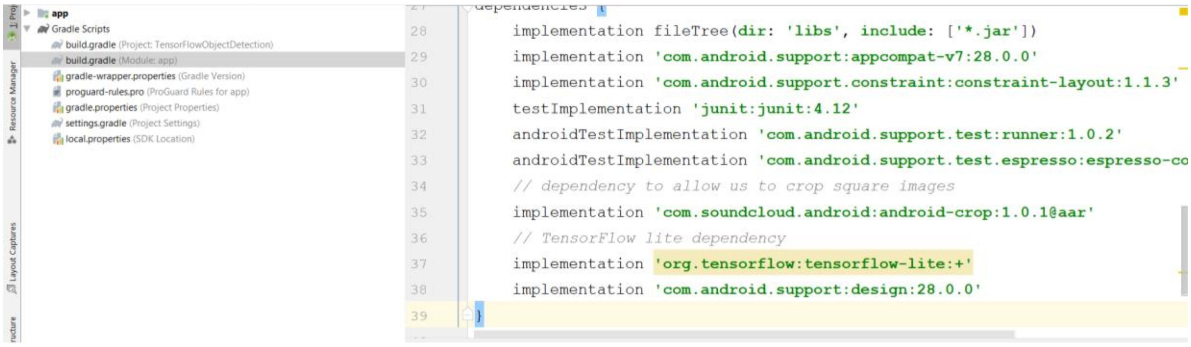
Documentation

Introduction:

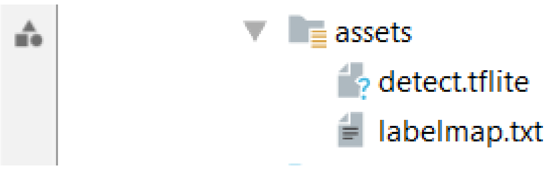
This is an object detection app created using Android Studio and TensorFlow Lite. The ML model used for this application is Coco SSD MobileNet (https://www.tensorflow.org/lite/models/object_detection/overview). This model is trained on 80 classes.

Import TensorFlow Lite Dependency in build.gradle (Module: App) file.


app
Gradle Scripts
build.gradle (Project: TensorFlowObjectDetection)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
1: Project
Resource Manager
Layout Captures
Structure

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-co
    // dependency to allow us to crop square images
    implementation 'com.soundcloud.android:android-crop:1.0.1@aar'
    // TensorFlow lite dependency
    implementation 'org.tensorflow:tensorflow-lite:+'
    implementation 'com.android.support:design:28.0.0'
}
```

Download the .zip file of the model from the link above and extract the detect.tflite and labelmap.txt to the assets folder.



There are 2 scripts in this project.

ChooseModel.java

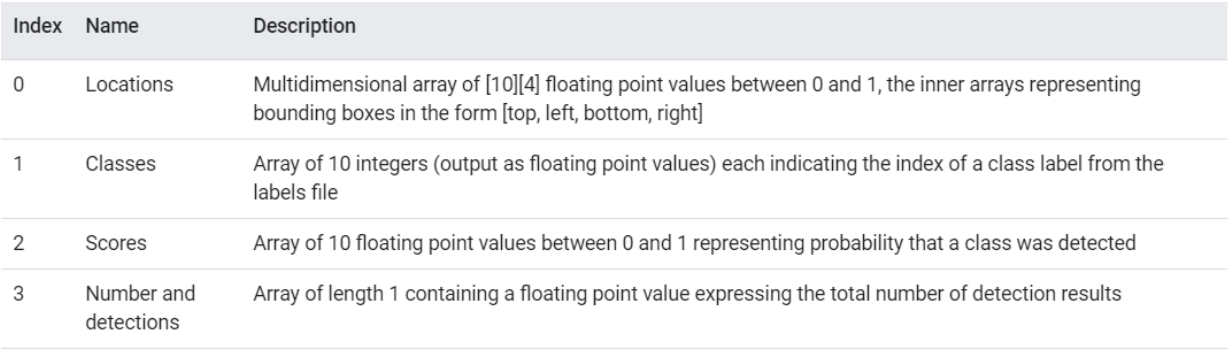
Classify.java

ChooseModel.java

This script does all the task from choosing the model to asking for different permission required for this app. It provides a camera interface to take the picture of the scene the user wants to detect objects in. The camera clicks the picture and passes to next script.

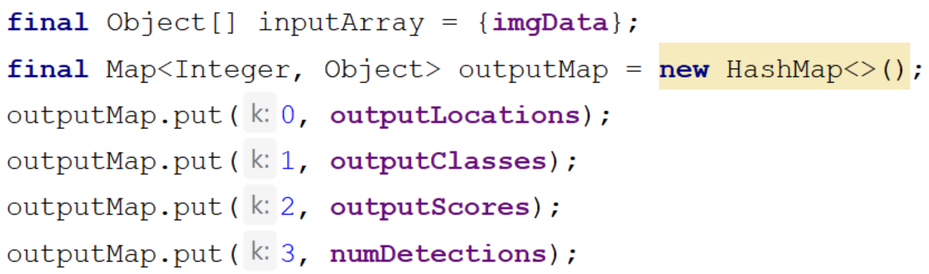
Classify.java

This is the core script for this app. The main part of system i.e. detection happens through this script. In this script, the clicked image is converted into bitmap format which is then changed to byte format as the quantized detection model requires so. The model requires an image of size 300x300. So, we also resize the image. This model provides output in the following format:

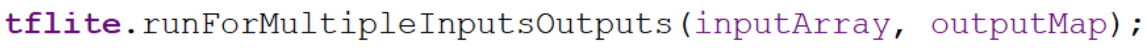


Index	Name	Description
0	Locations	Multidimensional array of [10][4] floating point values between 0 and 1, the inner arrays representing bounding boxes in the form [top, left, bottom, right]
1	Classes	Array of 10 integers (output as floating point values) each indicating the index of a class label from the labels file
2	Scores	Array of 10 floating point values between 0 and 1 representing probability that a class was detected
3	Number and detections	Array of length 1 containing a floating point value expressing the total number of detection results

As the model gives multiple output, we have to create a hash map of the output arrays to feed into model and use runForMultipleInputsOutputs attribute of the interpreted model.

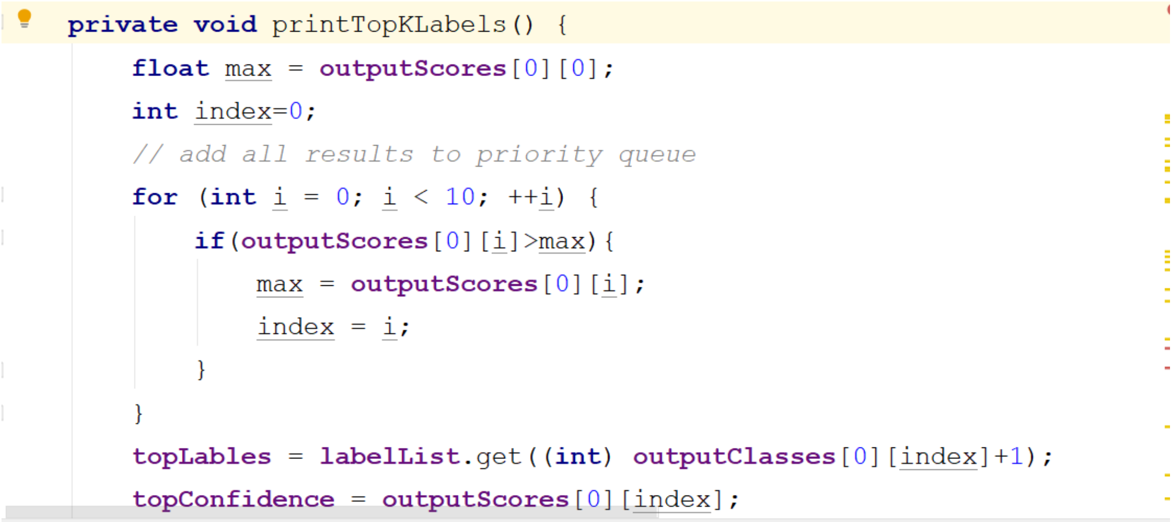


```
final Object[] inputArray = {imgData};
final Map<Integer, Object> outputMap = new HashMap<>();
outputMap.put( k: 0, outputLocations);
outputMap.put( k: 1, outputClasses);
outputMap.put( k: 2, outputScores);
outputMap.put( k: 3, numDetections);
```



```
tflite.runForMultipleInputsOutputs(inputArray, outputMap);
```

After the model gives it’s prediction, we get the class with highest confidence and print it’s label name and it’s confidence score.



```java
private void printTopKLabels() {
    float max = outputScores[0][0];
    int index=0;
    // add all results to priority queue
    for (int i = 0; i < 10; ++i) {
        if(outputScores[0][i]>max){
            max = outputScores[0][i];
            index = i;
        }
    }
    topLables = labelList.get((int) outputClasses[0][index]+1);
    topConfidence = outputScores[0][index];
```

Note: We have to increase the index of label by 1 as the counting in the label starts from 1 where as the index starts from 0.