Firebase ML Kit

Firebase is a big platform which helps to add advanced solutions for mobile apps, like push messaging, cloud database, machine learning solutions (ML Kit) and many others.

ML Kit is a mobile machine learning system. Main features including optical character recognition, detecting faces, scanning barcodes, labelling images and recognizing landmarks.

Project configuration

Before you can add Firebase to your Android app, you need to create a Firebase project to connect to your Android app.

1. Go to: https://console.firebase.google.com/
2. Click add project and provide it name
3. In the project view select android icon
4. Enter app's package name (applicationID form manifest.xml) in the Android package name field.
5. Optionally provide other app information: App nickname and Debug signing certificate SHA-1.
6. Click register app
7. Download google-services.json to obtain your Firebase Android config file and move it file into the module (app-level) directory of your app. Typically it is app directory.
8. In the root-level (project-level) Gradle file add:

```
buildscript {

  repositories {
    google()
  }

  dependencies {
    // ...
    classpath 'com.google.gms:google-services:4.3.3'
  }
}

allprojects {
  // ...

  repositories {
    google()
    // ...
  }
}
```

   Remember to update google-services to latest version.
9. Add the dependencies for the ML Kit Android libraries to your module (app-level):

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'

dependencies {
```

```
  // ...

  implementation 'com.google.firebase:firebase-ml-
vision:24.0.1'
  implementation 'com.google.firebase:firebase-ml-vision-
image-label-model:19.0.0'
}
```

Our example app will be labelling the photos that why we selected the vision module only.

10. Add the following declaration to app's AndroidManifest.xml file:

```
<application ...>
  ...
  <meta-data

android:name="com.google.firebase.ml.vision.DEPENDENCIES"
      android:value="label" />
</application>
```

11. Sync your gradle file.

Implementation

Right now, we can start the implementation of the app:

1. Create layout with ImageView (to show image), TextView (to show labels) and Button (to pick image)
2. To pick image form gallery add this code to your Activity class:

```
private fun pickImage() {
    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "image/*"
    startActivityForResult(intent, pickPhotoRequestCode)
}
```

It calls action get content for images which should open standard gallery application or allow user to select apps which support this intent.
`pickPhotoRequestCode` is an integer which identify our callback in onActivityResult method, suggest to use values above 100.

3. Connect this code with onClick for a button:

```
button.setOnClickListener { view ->
            pickImage()
}
```

Try run the app and click the button it should open the gallery app to pick a photo.

4. The next step is to implement the `onActivityResult` method to process the image:

```
override fun onActivityResult(requestCode: Int,
resultCode: Int, data: Intent?) {
        if (resultCode == Activity.RESULT_OK) {
            when (requestCode) {
                pickPhotoRequestCode -> {
                    val bitmap = getImageFromData(data)
                    bitmap?.apply {
                        contentIV.setImageBitmap(this)
                        processImageTagging(bitmap)
                    }
                }
            }
        }
        super.onActivityResult(requestCode, resultCode,
data)
}
```

`contentive` – is an id of ImageView
`getImageFromData` and `processImageTagging` are methods which we will implemented in next points. As you can see, we use `pickPhotoRequestCode` to define logic related to image returned by gallery.

5. Logic related to `getImageFromData`:

```
private fun getImageFromData(data: Intent?): Bitmap? {
        val selectedImage = data?.data
        return
MediaStore.Images.Media.getBitmap(this.contentResolver,
selectedImage)
}
```

6. The most important method is `processImageTagging` which uses ML Kit:

```
private fun processImageTagging(bitmap: Bitmap) {
        val visionImg =
FirebaseVisionImage.fromBitmap(bitmap)
        val labeler =
FirebaseVision.getInstance().getOnDeviceImageLabeler().
processImage(visionImg)
            .addOnSuccessListener { tags ->
                tagsTV.text = tags.joinToString(" ") {
it.text }
            }
            .addOnFailureListener { ex ->
                Log.wtf("LAB", ex)
            }
}
```

The line:
`val visionImg = FirebaseVisionImage.fromBitmap(bitmap)`
converts bitmap to FirebaseVissionImage which can be use by ML kit.

`FirebaseVision` introduce two methods to labeling (tagging) photos:

- getOnDeviceImageLabeler() process image directly on device using ready machine learning solutions from TensorFlow Lite.
- getCloudImageLabeler() process image in the cloud (required additional configuration)

Exercise:

Based on above implementation add functionality to:

a) (3 p.) process images from camera
b) (3 p.) add text recognition functionality and print text in similar way as tags in samples app
c) (4 p.) add object detector functionality and draw them on the image

It is recommended to use methods for on device processing.