Barcode

Le **BarCode API** sono specializzate nell'acquisizione dei dati memorizzati in forma di **codici a barre** di vari formati. Questi ultimi sono presenti in molti oggetti quotidiani, e vengono spesso utilizzati in applicazioni di natura logistica e commerciale.

Le Barcode API sono compatibili con tutti i formati più comuni di codici a barre come gli *EAN-13*, codici formati da 13 cifre. Sempre più utilizzati sono inoltre i **codici QR**, di natura bidimensionale.

I codici da interpretare possono essere rilevati in vari modi – da immagini, da foto scattate o da riprese effettuate con la fotocamera – ma essenzialmente i passi da percorrere sono, grosso modo, sempre gli stessi:

- istanzieremo un **BarcodeDetector**, che inizializzeremo specificando quali tipi di codici siamo interessati a rilevare: ogni codice è riconoscibile mediante le costanti raccolte nella classe Barcode;
- controlleremo che il detector sia utilizzabile tramite il metodo isOperational, altrimenti non potremo procedere;
- predisporremo la sorgente dati definendo le immagini da utilizzare o stabilendo connessioni con l'hardware che ci interessa;
- elaboreremo i risultati ottenuti via via verificando il contenuto degli oggetti BarCode restituiti.

Per poter svolgere il nostro esempio dovremo, per prima cosa, impostare alcuni aspetti della configurazione del progetto:

- nel file **build.gradle** del modulo applicativo richiediamo l'utilizzo dei Google Play Services, o almeno la porzione relativa alle API di Mobile Vision. Sarà necessario, al momento dell'utilizzo, indicare la versione dei Google Play Services che si desidera integrare, purchè questa non sia inferiore alla 7.8;
- nel file **AndroidManifest.xml** inseriremo un campo meta-data che servirà ad ottenere le dipendenze necessarie all'interpretazione dei codici a barre, e la permission relativa all'utilizzo della fotocamera di cui avremo bisogno

Il layout della nostra applicazione è costituito da una SurfaceView (la superficie in cui proietteremo quanto rilevato dalla fotocamera in tempo reale), una TextView con sfondo celeste in cui vedremo apparire i codici di volta in volta rilevati dal Detector, ed un pulsante con il semplice compito di pulire il contenuto della TextView. L'immagine mostra l'app in funzione: nella `SurfaceView` vediamo inquadrato un codice a barre presente su una bottiglia e la `TextView` sottostante dimostra come sia stato correttamente riconosciuto.

Il funzionamento ruota attorno a tre componenti: la **SurfaceView**, il **BarcodeDetector** ed un oggetto **CameraSource**, altra classe offerta da Vision che gestisce la fotocamera congiuntamente al BarcodeDetector. Li inizializziamo tutti nel metodo `onCreate`.

Il metodo receiveDetections viene definito in un Processor ed è il punto in cui vengono fornite le informazioni decodificate dal Detector. Se lo SparseArray restituito ha dimensione maggiore di zero, significa che abbiamo riscontrato almeno un risultato. Si noti che receiveDetections lavora su un thread secondario e, per questo motivo, aggiorniamo la TextView utilizzando il metodo runOnUiThread, con cui impacchettiamo operazioni da svolgere sul main thread, ove viene elaborata l'interazione con l'interfaccia utente.

L'attivazione della fotocamera avviene nel nostro metodo activateCamera, in cui gestiamo le permission in modalità dinamica visto che l'autorizzazione necessaria che ci attendiamo dall'utente è tra quelle considerate *dangerous*, ossia potenzialmente nocive per la privacy dell'utente.