REPORT ATTIVITÀ PROGETTUALE

SISTEMI DIGITALI

Giorgio Mocci e Miro Daniel Rajer

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Mocci Giorgio
Miro Daniel Rajer

Mocci Giorgio
Miro Daniel Rajer

Sommario

Mocci Giorgio
Miro Daniel Rajer

Introduzione

Questo progetto prevede la realizzazione di un applicativo Android che sfruttando una rete neurale sia in grado di riconoscere il volto dell'utente. Il sistema sfrutterà un modello di rete neurale addestrato per poter riconoscere il viso dell'utente in foto dove sono presenti anche più persone. L'applicativo permetterà sia di svolgere un filtraggio "live", cioè aprendo la camera dello smartphone e individuando *real-time* la faccia dell'utente, sia uno "statico" in cui si selezionano dalla galleria alcune foto e si ottiene in output l'elenco delle sole foto filtrate, cioè quelle in cui appare l'utente.

Tale caso di studio è un classico esempio di applicazione di machine learning e il software farà ricorso a una **rete neurale convoluzionale** (CNN). Tale scelta è dovuta al fatto che una rete neurale rappresenta il modo più comodo e pratico per problemi di classificazione, come quello di questa attività.

L'applicativo, inoltre, è pensato per la piattaforma Android e quindi tale progetto pone attenzione anche all'uso di risorse in quanto dovrà funzionare su smartphone, ovvero dispositivi embedded.

Per affrontare la complessità di questa attività si divide il lavoro da svolgere in sotto-progetti, ognuno dei quali con il compito di dare in uscita un prodotto a sé stante. La divisione in sotto-progetti è la seguente:

- **Rete**
  Partendo da un modello già esistente si procede con l'addestramento per ottenere in output un modello ad hoc e pronto all'uso.
- **Detector statico con Python**
  Partendo dal modello precedente si realizza in Android un detector statico implementato in Python e Java.
- **Detector live**
  Realizzazione di un riconoscitore di volti in real-time in Android partendo dal modello addestrato.
- **Detector statico senza Python**
  Revisione più efficiente del detector statico realizzato per Android solo in linguaggio Java.
- **Realizzazione dell'app**
  Unione dei vari prodotti per realizzare un'applicazione Android completa user-friendly.

Data un'introduzione al progetto si passa ora a descrivere le reti neurali coinvolte.

Mocci Giorgio
Miro Daniel Rajer

Rete neurale

L'obbiettivo primario di questo task è produrre un modello di rete neurale addestrata in grado di riconoscere un volto dato. A tal scopo si è utilizzato un modello messo a disposizione dalla libreria TensorFlow ottimizzato per il training di modelli specifici per la classificazione di immagini. Visto l'ambito dei sistemi embedded nel quale il progetto complessivo si colloca si è optato per MobileNet, una CNN pensata per dispositivi mobile. MobileNet è il primo modello di computer vision pensato per dispositivi embedded basato su TensorFlow. MobileNet è sufficientemente leggera e veloce da essere eseguita su smartphone senza consumo di risorse eccesivo mantenendo comunque una precisione adeguata.

Per il training del modello si prevedono due classi: "*me*", per le foto che contengono il viso cercato, e "*not me*" per le altre. Per la categoria "not me" si è usato Flickr-Faces-HQ Dataset (FFHQ), un dataset di volti con licenza Creative Commons 2.0. Per la categoria "me" nel caso d'uso previsto spetterà all'utente specificare proprie foto. A fini di testing si è preso ad esempio il volto di un personaggio pubblico, quello di Donald Trump. Si è quindi proceduto all'addestramento della rete tramite Python usando TensorFlow e le API di Keras. Si è ottenuto in output un modello addestrato e pronto all'uso, che poi è stato convertito in un formato adatto ai sistemi embedded (TensorFlow Lite).

Ai fini di addestrare al meglio la rete neurale e ottenere poi risultati migliori, si è deciso di utilizzare una seconda rete neurale esistente atta al riconoscimento dei volti presenti in una foto. Questo permetterà alla precedente rete di analizzare tutti i volti presenti nella stessa immagine separatamente. Nello specifico, si è utilizzata la rete MTCNN (Multi-task Cascaded Convolutional Networks) nell'addestramento e nel detector statico basato su Python. Mentre invece le API Vision di Google ML Kit per i detector basati su Android.

Queste reti neurali permettono di procedere con il prossimo task, ovvero la realizzazione di un detector statico.

Mocci Giorgio
Miro Daniel Rajer

Sviluppo su Android

Detector statico con Chaquopy

Dopo aver esposto le reti neurali coinvolte in questo progetto si passa a descrivere l'implementazione svolta. La piattaforma scelta è quella di Android e il linguaggio è Java.

Si è iniziato dal detector statico. Il compito del detector statico è quello di prendere in input delle immagini selezionate dalla galleria e dare in output le sole foto in cui compare il volto dell'utente riconosciuto, scartando le altre. Come prima soluzione si è scelto di far girare il modello della rete neurale ottenuto precedentemente (`model.tflite`) tramite Python.

Con Python è possibile caricare un modello di TensorFlow Lite da usare come interprete per il filtraggio delle foto.

```
# Load model to face classification
nomeFile = join(dirname(__file__), "model.tflite")
interpreter = tflite.Interpreter(model_path=nomeFile)
interpreter.allocate_tensors()
class_names = ['me', 'not_me']
```

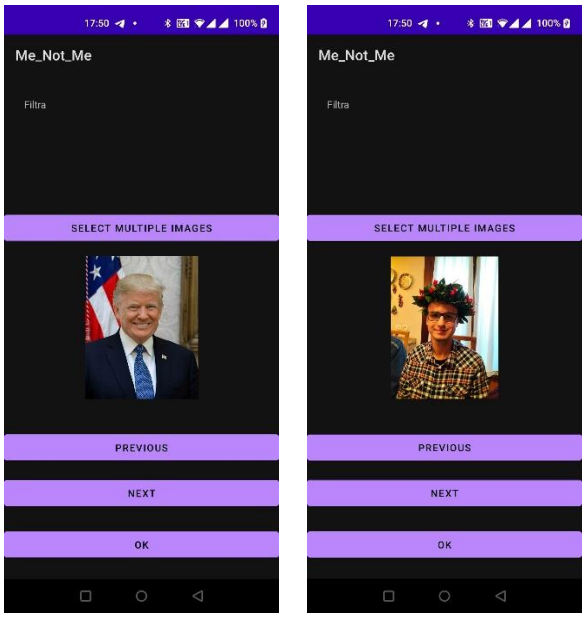
Si è quindi realizzato uno script Python che data un'immagine è in grado dire se vi è presente il volto dell'utente. Si crea quindi un detector con MTCNN. Tale detector data un'immagine dà in output l'elenco delle posizioni delle facce trovate. Ognuna di questa viene passata all'interprete di TensorFlow Lite. Se tra tutte le facce vi è quella dell'utente allora l'immagine iniziale va conservata, altrimenti scartata.

Perciò ora si ha uno script Python, chiamato `detector.py`, che permette di filtrare le immagini. Per poterlo usare all'interno di Java si adotta l'uso di **Chaquopy**. Chaquopy è un tool per l'integrazione di Python in progetti Android. Mette a disposizione API per chiamare codice Python da Java e/o Kotlin.

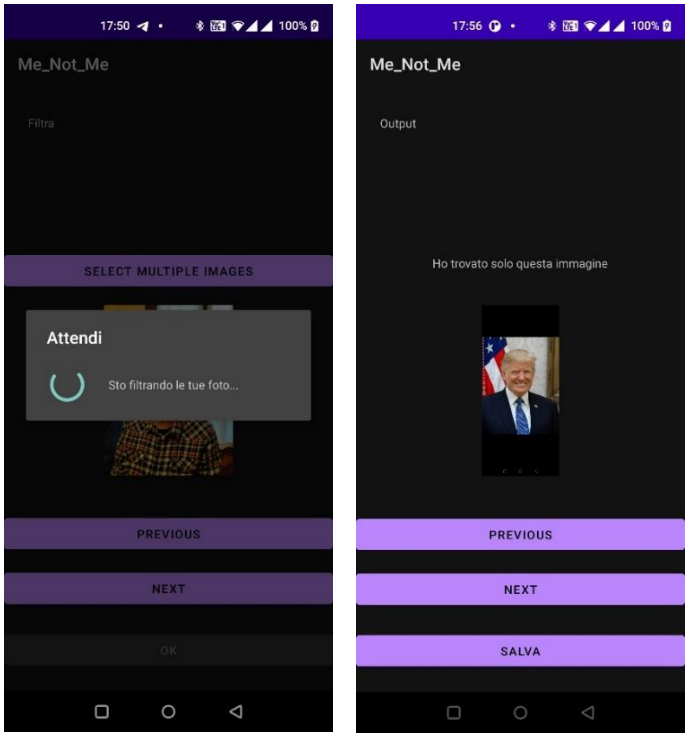
```
//ottenimento di un PyObject che rappresenta la funzione python da chiamare
//all'interno dello script detector.py
PyObject obj = Python.getInstance().getModule( name: "detector").callAttr( key: "main");
```

Si implementa quindi una app in Android scritta in Java che realizza la funzionalità di filtraggio statico delle foto basandosi sul detector scritto in Python.

L'app permette di selezionare le foto dalla galleria del telefono. Dopodiché mostra le foto selezionate dall'utente, il quale poi può avviare il filtraggio. Una volta completato il filtraggio una apposita schermata mostra le foto in output, con la possibilità di salvarle una per una in una cartella apposita del dispositivo. Si mostrano alcuni screenshot d'esempio.



Vengono selezionate due foto d'esempio. Tali foto vengono mostrate una per volta. I pulsanti "*previous*" e "*next*" permettono di scorrere le foto selezionate per poterle vedere tutte. Il pulsante "*ok*" avvia il filtraggio. Una volta completata il filtraggio in output si hanno le sole foto che combaciano, in questo esempio una. Il pulsante "*salva*" permette di salvare l'immagine che si sta visualizzando nella cartella Pictures.



Chaquopy ha due grossi svantaggi. Il primo è che essendo un prodotto con licenza per poter distribuire l'app è necessario acquistare tale licenza. Se non lo si fa, oltre all'impossibilità materiale e legale di distribuire l'app, si incorre in una limitazione sul tempo di utilizzo del codice Python. Se l'esecuzione del codice Python impiega un tempo superiore a tale limite l'app viene chiusa forzatamente. Il secondo problema riguarda il consumo di risorse. L'uso di Chaquopy impedisce l'accesso a ottimizzazioni hardware inficiando molto negativamente sulle prestazioni. Essendo questo applicativo pensato per dispositivi embedded il fattore prestazionale diventa centrale. Allo stato

attuale anche con poche immagini il tempo di filtraggio diventa davvero lungo, costituendo un problema per l'utente dello smartphone.

Ciò inoltre rende Chaquopy praticamente impossibile da usare per il prossimo step del progetto, ovvero la realizzazione del detector in tempo reale, dove c'è bisogno di una maggiore reattività. Difatti, per il detector live si è proceduto usando direttamente le librerie Tensorflow per Java. Successivamente, alla luce di ciò si rivedrà anche l'implementazione del detector statico. Si procede ora a descrivere l'implementazione del detector live.

Mocci Giorgio
Miro Daniel Rajer

Detector Live

Dopo aver descritto l'implementazione del detector statico con Chaqoupy si passa ora a delineare lo step successivo del progetto ovvero il detector live. Mentre il detector statico funziona con foto già esistenti nella galleria del telefono, il detector live offre una classificazione in tempo reale tramite la fotocamera.

Il precedente task utilizza Chaquopy per chiamare codice Python contenente TensorFlow e le reti neurali. Tale soluzione risulta impraticabile per le motivazioni già esposte in precedenza. Una valida alternativa è sfruttare le apposite librerie di Tensorflow scritte in Java che permettono di usare il modello addestrato usando al meglio l'hardware del dispositivo. Tale soluzione viene usata per la realizzazione del detector live di cui ora si delineano i dettagli implementativi.

Per realizzare il detector live è necessario accedere alla camera del telefono e ciò è possibile tramite le API di CameraX.

```
private void startCamera() {

    final ListenableFuture<ProcessCameraProvider> cameraProviderFuture = ProcessCameraProvider.getInstance( context: this);

    cameraProviderFuture.addListener(new Runnable() {
        @Override
        public void run() {
            try {

                ProcessCameraProvider cameraProvider = cameraProviderFuture.get();
                bindPreview(cameraProvider);


            } catch (ExecutionException | InterruptedException e) {
                // No errors need to be handled for this Future.
                // This should never be reached.
            }
        }
    }, ContextCompat.getMainExecutor( context: this));
}
```

Per realizzare il filtraggio è necessario mandare i frame presi dalla camera ad un analizzatore di immagini. Ciò è possibile tramite la classe `ImageAnalysis`.

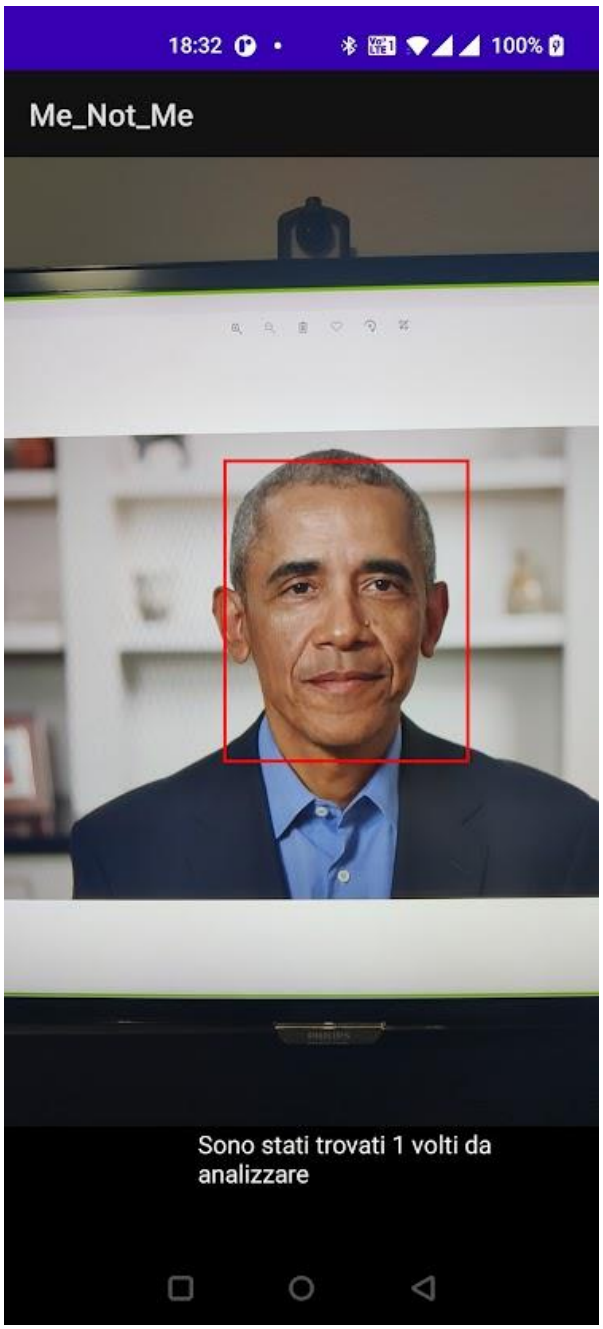
```
ImageAnalysis imageAnalysis = new ImageAnalysis.Builder()
        .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
        .build();
```

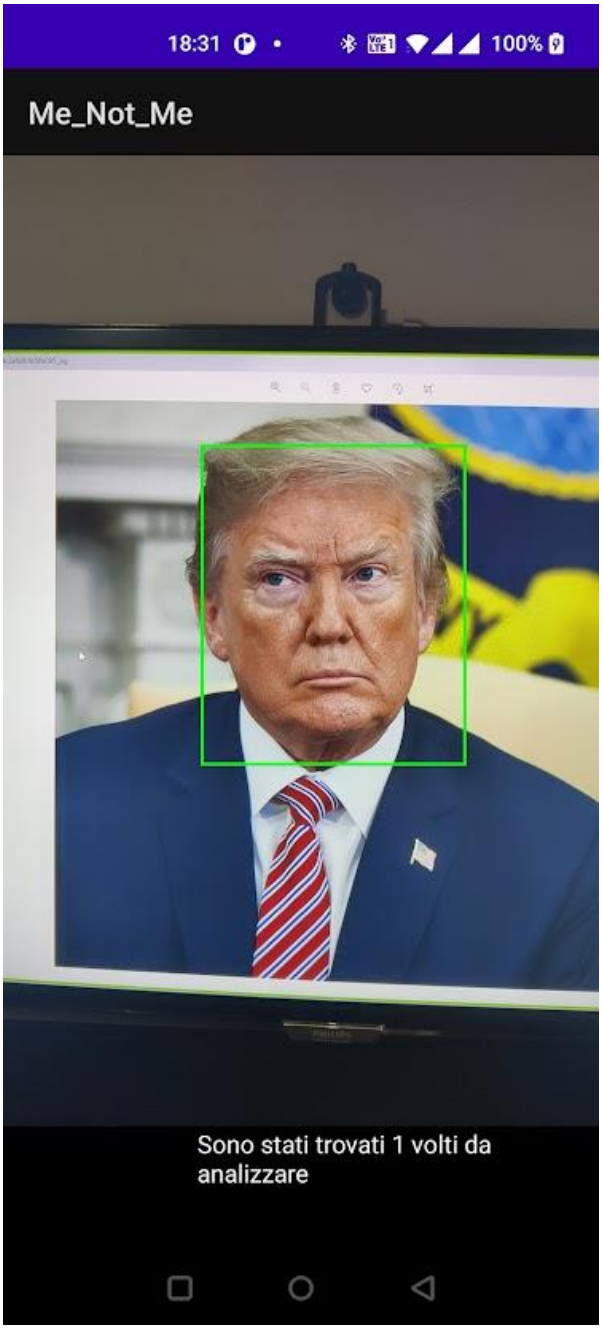
Tramite il metodo `imageAnalysis.setAnalyzer` è possibile specificare come analizzare ogni frame tramite un oggetto `ImageAnalysis.Analizer` impostato ad hoc.

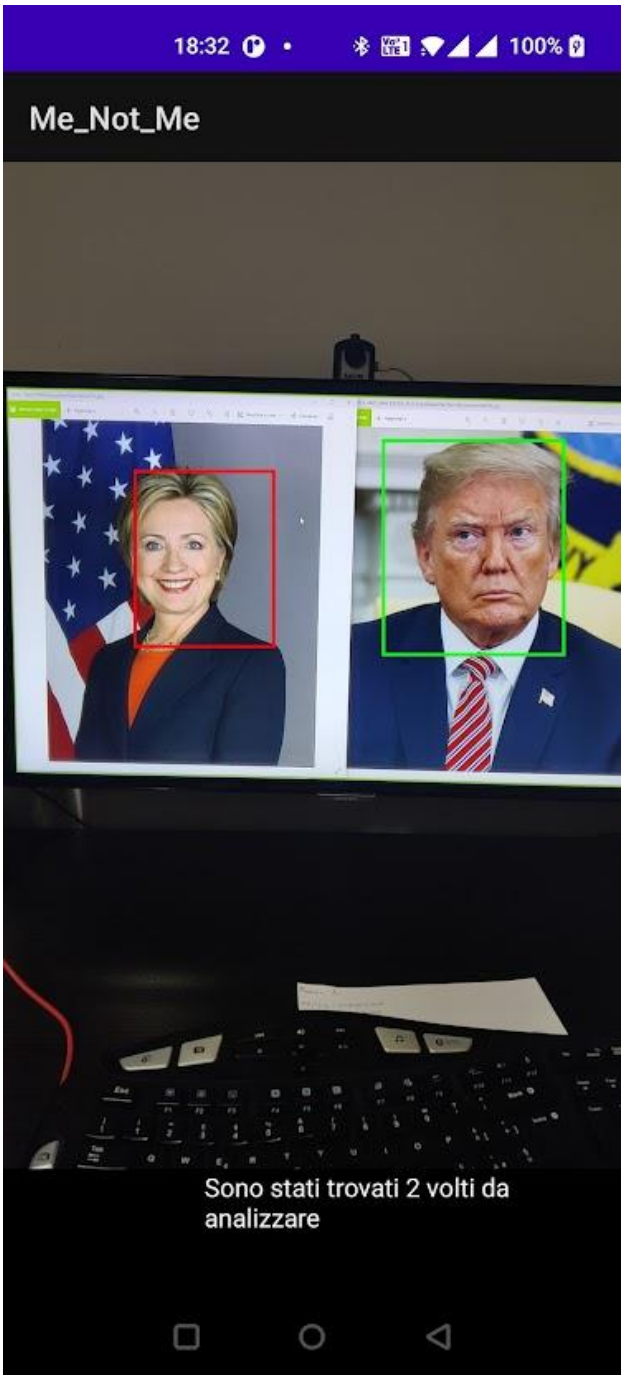
All'interno dell'`Analizer` sfruttando le API di `Google ML Kit`, si ottiene l'elenco di tutte le facce presenti nel frame. Ogni faccia viene passata all'interprete di TensorFlow Lite contenente il modello addestrato. Successivamente si esegue l'interprete tramite il metodo `run`. Se la faccia viene riconosciuta si disegna a video un rettangolo verde attorno ad essa, altrimenti un rettangolo rosso.

```
MappedByteBuffer tfliteModel = FileUtil.loadMappedFile(getApplicationContext(), filePath: "model.tflite");
Interpreter tflite = new Interpreter(tfliteModel);
```

Mocci Giorgio
Miro Daniel Rajer







Dopo aver descritto il detector live si espone la revisione del detector statico.

Detector statico ottimizzato

Precedentemente è stata realizzata una versione del detector statico basata su Python e Chaquopy. Per i problemi delineati sopra (inerenti a licenza e prestazioni) si è deciso di rivedere il progetto del detector statico usando le librerie di TensorFlow scritte in Java per accedere alle ottimizzazioni hardware, le stesse usate nel progetto del detector live.

Il caso d'uso è il medesimo di prima: l'utente seleziona le foto e l'app le filtra. A cambiare è solo la parte *core*, quella che realizza il filtraggio.

```
//caricamento del modello tflite
MappedByteBuffer tfliteModel = FileUtil.loadMappedFile(getApplicationContext(), filePath: "model.tflite");
//inizializzazione dell'interprete
Interpreter tflite = new Interpreter(tfliteModel);
```

```
//per ogni foto selezionata si ricavano tutte le facce presenti in essa
resultFace = detector.process(InputImage.fromBitmap(bitmapFOTO, rotationDegrees: 0));
```

Analogamente al detector live, con il metodo `run` dell'interprete TensorFlow si ottiene il risultato della rete neurale.

```
resized = Bitmap.createScaledBitmap(croppedBitmap, dstWidth: 250, dstHeight: 250, filter: false);
int imageTensorIndex = 0;
DataType imageDataType = tflite.getInputTensor(imageTensorIndex).dataType();
TensorImage tfImage = new TensorImage(imageDataType);
tfImage.load(resized);
float[][] labelProbArray = new float[1][2];
tflite.run(tfImage.getBuffer(), labelProbArray);
```

Mocci Giorgio
Miro Daniel Rajer

Realizzazione dell'app

Dopo aver concluso l'ultimo task si passa a unire tutto in un'unica applicazione Android.

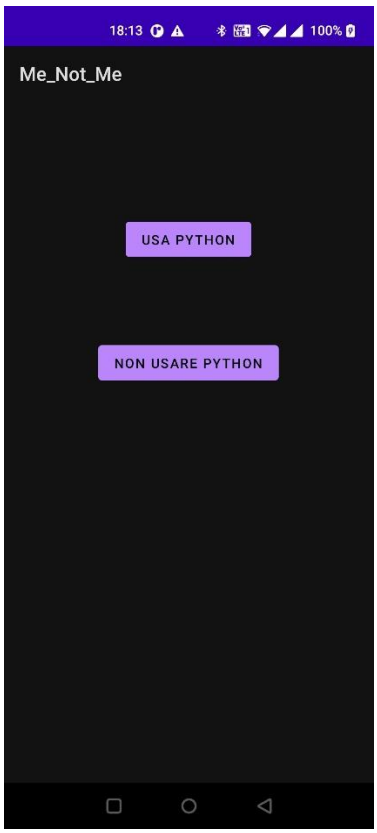
La prima schermata chiede all'utente di scegliere tra la parte con Chaquopy e quella senza.

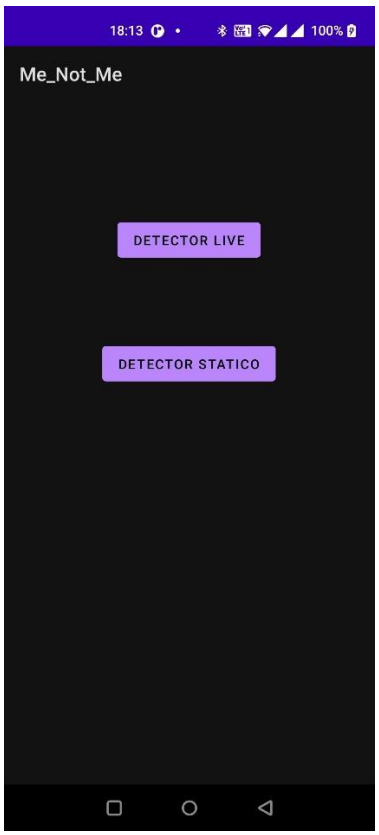
Selezionando la parte con Chaquopy si apre la schermata che permette di selezionare le immagini per il filtraggio statico.

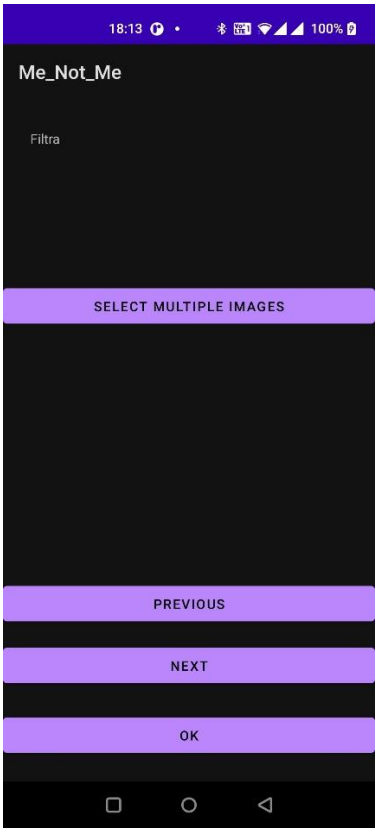
Selezionando la parte senza Chaquopy si apre una schermata che permette di scegliere tra il detector live e quello statico.

Il detector live apre la camera del telefono e inizia ad analizzare i frame di quest'ultima. Successivamente disegna un rettangolo verde attorno ai volti riconosciuti e uno rosso attorno agli altri.

Il detector statico di questa sezione è quello ottimizzato. Offre le medesime funzionalità dell'altra versione sfruttando l'implementazione più efficiente.







Mocci Giorgio
Miro Daniel Rajer

Conclusioni

Gli obiettivi del progetto sono stati raggiunti a pieno e con risultati soddisfacenti. L'applicativo è conforme alle aspettative e svolge i compiti adeguatamente.

La tecnologia di Chaquopy si è rivelata inadeguata per i sistemi embedded. Invece, le parti realizzate in Android risultano con prestazione adeguate.

Il modello addestrato restituisce un output corretto la maggior parte delle volte.