Indice

1 — Introduzione MediaPipe

L'applicazione sfrutta le potenzialità offerte dalla libreria `MediaPipe`, che utilizza soluzioni basate su Machine Learning nel contesto dello streaming multimediale. In particolare, noi abbiamo sfruttato la soluzione per il riconoscimento del movimento delle mani: **MediaPipe Hands**.

Il nostro scopo è stato quello di interpretare ed associare movimenti delle mani a determinate gesture in modo tale da poter gestire la lettura di un documento testuale, come un pdf.
Le funzionalità che abbiamo scelto di implementare sono:

- zoom
- scorrimento verticale e laterale delle pagine
- screenshot

Abbiamo deciso di realizzare l'applicazione per ambiente Android, tramite l'utilizzo di Java ed Android Studio.

1.1 MediaPipe Hands

MediaPipe Hands offre funzionalità per il tracciamento delle mani e delle dita, utilizzando il ML per individuare 21 punti di riferimento a 3 dimensioni di una mano da un solo fotogramma.
Viene sfruttata una pipeline composta da due modelli che lavorano insieme:

1. un modello per il **rilevamento del palmo** che opera sull'immagine completa e restituisce un riquadro di delimitazione orientato (*ritaglio*) della mano.
2. un modello che opera sulla regione dell'immagine ritagliata definita dal modello precedente e restituisce i **punti di riferimento della mano** a 3 dimensioni.

Il fatto di fornire al modello per i punti di riferimento un'immagine della mano accuratamente ritagliata diminuisce di gran lunga le elaborazioni necessarie, aumentando

l'efficienza.
Inoltre, nella pipeline i *ritagli* della mano possono essere generati anche in base ai punti di riferimento identificati nel frame precedente, e solo quando il modello del punto di riferimento non è più in grado di identificare la presenza della mano viene invocato il modello di rilevamento del palmo.

Specifica delle pipeline

La pipeline è implementata come un **grafo** che utilizza un **sottografo per il tracking dei punti** dal modulo `landmark` e renderizza usando un sottografo dedicato al rendering.
Il sottografo per il tracking dei punti utilizza al suo interno un'altro sottografo per il tracking e un **sottografo per il rilevamento della mano** dal modulo `palm detection`.

1.1.1 Modelli

Palm Detection Model

Per la rilevazione delle posizione iniziali delle mani, viene adottato un modello di rilevamento *single-shot* ottimizzato per l'utilizzo in tempo reale.
La rilevazione delle mani è però un compito complesso per via della loro omogeneità visiva, a differenza per esempio del viso, che invece ha dei pattern a più alto contrasto come nelle zone degli occhi e della bocca. Viene utilizzato perciò un contesto aggiuntivo, come possono essere la posizione del braccio o del corpo di una persona, in modo tale da poter individuare la mano più facilmente, aumentando l'efficienza.

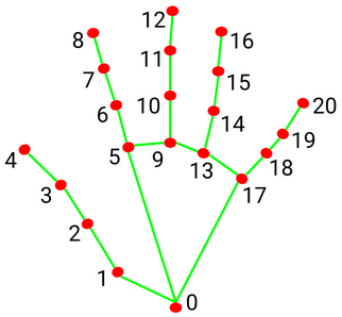
Il primo componente utilizzato è un **rilevatore del palmo**, piuttosto che della mano, in quanto *ritagli* di oggetti rigidi come palmi e pugni sono nettamente più semplici rispetto a quelli delle dita.
In secondo luogo, viene utilizzato un de/codificatore per l'estrazione di caratteristiche dall'immagine in modo da avere una maggiore consapevolezza del contesto della scena, anche per piccoli oggetti.
Infine, la perdita focale viene ridotta al minimo durante l'allenamento per supportare una grande quantità di ancoraggi derivanti dalla varianza su scala elevata.

Viene stimata una precisione di circa il 96% nella rilevazione del palmo.

Hand Landmark Model

Dopo il rilevamento del palmo, questo modello esegue, all'interno delle regioni rilevate, la localizzazione precisa di 21 punti della mano tramite **regressione**, ovvero la previsione diretta delle coordinate.
Il modello apprende quindi una rappresentazione della posizione del'interno della mano coerente ed è robusto anche a mani parzialmente visibili.

Per arrivare a questo risultato sono state annotate manualmente circa 30 mila immagini dal mondo reale con 21 coordinate 3D (valore Z preso tramite mappa di profondità). E' stato inoltre utilizzato un modello di mano sintetica su vari sfondi, mappato alle corrispondenti coordinate, in modo tale da avere un accuratezza maggiore.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

1.1.2 API

1.1.3 Opzioni di configurazione

`STATIC_IMAGE_MODE` Se settata a `false` (default), le immagini in ingresso vengono trattate come un flusso video, cercando di rilevare le mani nelle prime immagini di input e localizzando poi i punti di riferimento della mano. Nelle immagini successive, una volta che tutte le mani `MAX_NUM_HANDS` vengono rilevate e i corrispondenti punti di riferimento della mano sono localizzati, si limita a tracciare quei punti di riferimento senza invocare un altro rilevamento fino a quando non perde traccia di una qualsiasi delle mani. Questo riduce la latenza ed è ideale per l'elaborazione di fotogrammi video.
Se impostato a `true`, il rilevamento manuale viene eseguito su ogni immagine di input, ideale per elaborare un batch di immagini statiche, magari non correlate.

`MAX_NUM_HANDS` Massimo numero di mani da rilevare (2 a default).

`MODEL_COMPLEXITY` Complessità del modello di landmark: 0 o 1 (default). La precisione del punto di riferimante e la latenza aumentano con la complessità.

`MIN_DETECTION_CONFIDENCE` Valore minimo di fiducia $[0.0, 1.0]$ nei confronti del modello di rilevamento della mano per considerare il rilevamento della mano corretto. A defult è 0.5.

`MIN_TRACKING_CONFIDENCE` Valore minimo di fiducia $[0.0, 1.0]$ nei confronti del modello di tracking per considerare il rilevamento dei punti di riferimento corretto. Altrimenti, il modello di rilevamento della mano (palmo) verrà richiamato all'immagine in input successiva. Di conseguenza, aumentando questo valore avremo una robustezza maggiore, a discapito di un aumento di latenza. A defult è 0.5.

Output

`MULTI_HAND_LANDMARKS` Raccolta di mani rilevate/tracciate, ognuna rappresentata come una lista di 21 punti di riferimento a 3 dimensioni. In ogni punto `x` e `y` sono normalizzati ($[0.0, 1.0]$) rispetto alla larghezza e all'altezza dell'immagine. La coordinata `z` rappresenta invece la profondità del punto rispetto a quella del polso (considerato l'origine). Minore è il valore della `z`, più il punto è considerato vicino alla fotocamera.

`MULTI_HAND_WORLD_LANDMARKS` Come il precedente, ma i punti sono valutati come coordinate 3D reali in metri, con l'origine al centro geometrico approssimativo della mano.

`MULTI_HAND_HANDEDNESS` Raccolta delle *handedness* della mani (destra/sinistra). Ogni mano è composta da un'etichetta (`label`) e da un punteggio (`score`). L'etichetta è una stringa di valore `left` o `right`, mentre il punteggio rappresenta la stima della probabilità di predizione corretta della mano (sempre ≥ 0.5, mentre l'*handedness* opposta è $1-$`score`). L'*handedness* viene valutata considerando che l'immagine in input è specchiata.

OpenGL ES

Per il rendering dell'input e dell'output viene utilizzato `OpenGL ES`, un sottoinsieme delle librerie grafiche di `OpenGL` pensato per dispositivi embedded.
Nella porzione di codice sottostante viene mostrato un esempio di utilizzo.

```
    // Vengono prima settate le opzioni di configrazione citate precedentemente
    HandsOptions handsOptions =
        HandsOptions.builder()
            .setStaticImageMode(false)
```

```
            .setMaxNumHands(2)
            .setRunOnGpu(true).build();
    Hands hands = new Hands(this, handsOptions);
    hands.setErrorListener(
        (message, e) -> Log.e(TAG, "MediaPipe Hands error:" + message));

    // Inizializzato una nuova istanza di CameraInput
    // e connessione alla soluzione MediaPipe Hands
    CameraInput cameraInput = new CameraInput(this);
    cameraInput.setNewFrameListener(
        textureFrame -> hands.send(textureFrame));

    // Inizializzato una istanza di GlSurfaceView tramite
    // ResultGlRenderer<HandsResult> che fornisce un interfaccia
    // per eseguire codice di rendering OpenGL definito dall'utente
    SolutionGlSurfaceView<HandsResult> glSurfaceView =
        new SolutionGlSurfaceView<>(
            this, hands.getGlContext(), hands.getGlMajorVersion());
    glSurfaceView.setSolutionResultRenderer(new HandsResultGlRenderer());
    glSurfaceView.setRenderInputImage(true);

    hands.setResultListener(
        handsResult -> {
        if (result.multiHandLandmarks().isEmpty()) {
            return;
        }
        NormalizedLandmark wristLandmark =
            handsResult.multiHandLandmarks().get(0).getLandmarkList()
                .get(HandLandmark.WRIST);
        Log.i(
            TAG,
            String.format(
                "MediaPipe Hand wrist normalized coordinates
                (value range: [0, 1]): x=%f, y=%f",
                wristLandmark.getX(), wristLandmark.getY()));
        // Richiesta di rendering a OpenGL
        glSurfaceView.setRenderData(handsResult);
        glSurfaceView.requestRender();
        });

    // Avvio della fotocamera dopo che GLSurfaceView si è connesso
    glSurfaceView.post(
        () ->
            cameraInput.start(
                this,
                hands.getGlContext(),
                CameraInput.CameraFacing.FRONT,
                glSurfaceView.getWidth(),
```

```
                glSurfaceView.getHeight()));
```

1.2 Struttura del progetto di partenza

Oltre a file ausiliari come il *manifest* o gli *xml* di configurazione, il cuore del progetto è rappresentato da 3 file nella cartella `hands/java/com/google/mediapipe/examples/hands`.

Ovviamente, `MainActivity` corrisponde all'entry point dell'applicazione, partiamo quindi dall'analisi di questo file.

1.2.1 MainActivity

In particolare, verrà analizzato il comportamento dell'applicazione quando viene preso l'input tramite streaming video dalla fotocamera. Sono presenti infatti 3 `InputSource` disponibili: `CAMERA`, `VIDEO` e `IMAGE`.

Vengono inizialmente create le istanze di:

- Hands
- CameraInput
- InputSource
- SolutionGlSurfaceView<HandsResult>

Successivamente abbiamo 3 metodi implementati con *override*, ereditati dall'estensione di `AppCompactActivity`, un modello predefinito fornito da Android Studio.

onCreate

Viene passato un *Bundle* in input, utilizzato per chiamare la `onCreate` della classe estesa. In Android Studio i *Bundle* si utilizzano solitamente per il passaggio di dati da un attività all'altra, in questo caso abbiamo un istanza di uno stato salvato precedentemente (e.g. l'applicazione era stata messa in background).
Viene poi settato il *ContentView* (ovvero il layout della schermata coi pulsanti) tramite il file `activity_main.xml`, che definisce appunto il layout.
Infine, viene chiamata la funzione `setupLiveDemoUiComponents()` per inizializzare i componenti per lo streaming video.

onResume

Dopo essere stata chiamata la `onResume` della classe estesa, si istanzia un nuovo `CameraInput`, passandogli l'*activity* corrente, per poi settare il *FrameListener*.
Successivamente viene utilizzata l'istanza di `SolutionGlSurfaceView<HandsResult>` per chiamare il metodo interno `post`, passandogli la funzione `startCamera()` (il cui scopo è quello di inizializzare lo streaming video). In questo modo, tramite la `post` ereditata dalla classe `View`, viene eseguita la funzione passata ed esposto l'output nell'interfaccia utente.

onPause

Dopo essere stata chiamata la `onPause` della classe estesa, viene nascosta la visibilità del render (`glSurfaceView`) e viene fermato il flusso video in input dalla fotocamera.

setupLiveDemoUiComponents

Questo metodo viene chiamato all'inizializzazione. Recupera il bottone per avviare lo streaming e ci aggancia un listener sul click. Se è già impostata la modalità di assunzione dell'input dalla fotocamera non fa nulla, altrimenti termina la pipeline corrente e chiama la `setupStreamingModePipeline(InputSource source)`, passando come `source` il valore di `InputSource.CAMERA`.

setupStreamingModePipeline

Questa funzione ha lo scopo di inizializzare un nuova pipeline per il rilevamento dell mani, differenziata in base al tipo di `InputSource`.
Per prima cosa, viene creata una nuova istanza di `HandsResultGlRenderer`

2 — …

3 — ...

4 — ...

5 — ...