

ALMA MATER STUDIORUM · UNIVERSITÀ DI
BOLOGNA

Corso di Laurea Magistrale in
Ingegneria Informatica

Covid Protection Checker

Progetto Sistemi Digitali M

Autori:

Cristian Davide Conte
0001034932

Simone Morelli
0001034742

Anno Accademico
2021-2022

Indice

1	Abstract	2
2	Reti Neurali	3
2.1	Tecnologie utilizzate	3
2.2	Dataset utilizzato	3
2.3	Architetture reti neurali	5
2.4	Training e risultati	8
3	Architettura Applicazione	10
3.1	Descrizione generale	10
3.2	MainActivity	10
3.3	CameraProvider	12
3.4	AnalyzeActivity	15
4	Problemi affrontati	18
4.1	Rotazione fisica della fotocamera del dispositivo	18
4.2	Data augmentation	19
5	Risultati ottenuti	21
6	Estensione del progetto	24
6.1	Abstract	24
6.2	Metodo di risoluzione	24
6.3	Problemi del metodo e risultati	27

Capitolo 1

Abstract

Ispirati dalla recente pandemia e dalla situazione eccezionale con cui siamo stati costretti a convivere, abbiamo realizzato un'applicazione Android che permette di rilevare e categorizzare il corretto utilizzo delle mascherine. Il sistema, laddove la mascherina non fosse indossata correttamente, sarebbe inoltre in grado di fornire dettagli circa l'errato posizionamento di quest'ultima. L'applicazione è inoltre capace di elaborare sia flussi di immagini che singoli frame.

Il codice sorgente è disponibile a questo *[link github](#)*.

Capitolo 2

Reti Neurali

2.1 Tecnologie utilizzate

L'applicazione è completamente realizzata in Java ed è supportata da Android 11+.

Per la sua realizzazione è stato necessario l'impiego di tecnologie quali: Android Studio, python3, Google Colab, API CameraX, API MediaStore, TensorFlow 2 Lite e API TFLiteModelMaker.

2.2 Dataset utilizzato

Al fine di effettuare la parte di object detection è stata impiegata una rete neurale convoluzionale allenata tramite il dataset *WWMR-DB* [1].

Quest'ultimo contiene circa **1200 foto** di volti scattate da varie inclinazioni e divise in **8 classi** che descrivono i modi più comuni di indossare una mascherina:

- mascherina indossata correttamente
- mascherina non indossata affatto
- mascherina pendente da un orecchio
- mascherina sulla punta del naso
- mascherina sulla fronte
- mascherina sotto al naso
- mascherina sopra al mento
- mascherina sotto al mento

I volti sono appartenenti a **42 soggetti** ed ognuno indossa uno tra i seguenti tipi di mascherina/respiratore:

- mascherina non medica
- mascherina chirurgica
- respiratore con valvola
- respiratore senza valvola

Ogni foto è associata ad un file di testo in formato YOLO (detto *file di label*) le cui righe descrivono le caratteristiche della bounding-box e della classe associata all'immagine.

<image-class> <box-center-x> <box-center-y> <box-width> <box-height>

Figura 2.1: Esempio di una riga di un file di label in formato YOLO

2.3 Architetture reti neurali

Poiché l'applicazione deve funzionare sia con una moltitudine di frame acquisiti in istanti ravvicinati nel tempo, sia in presenza di una singola immagine caricata direttamente dalla memoria del telefono, abbiamo ritenuto opportuno diversificare i due contesti e utilizzare due reti neurali differenti: EfficientDet-Lite 0 e EfficientDet-Lite 4 [2].

Dato che l'ambito è quello del riconoscimento e della classificazione di immagini, entrambe le reti neurali sono di tipo convoluzionale ma variano nello scaling della loro architettura.

Entrambe le CNN partono infatti dalla rete base EfficientNet [3], divisa per semplicità in moduli, realizzata nel seguente modo:

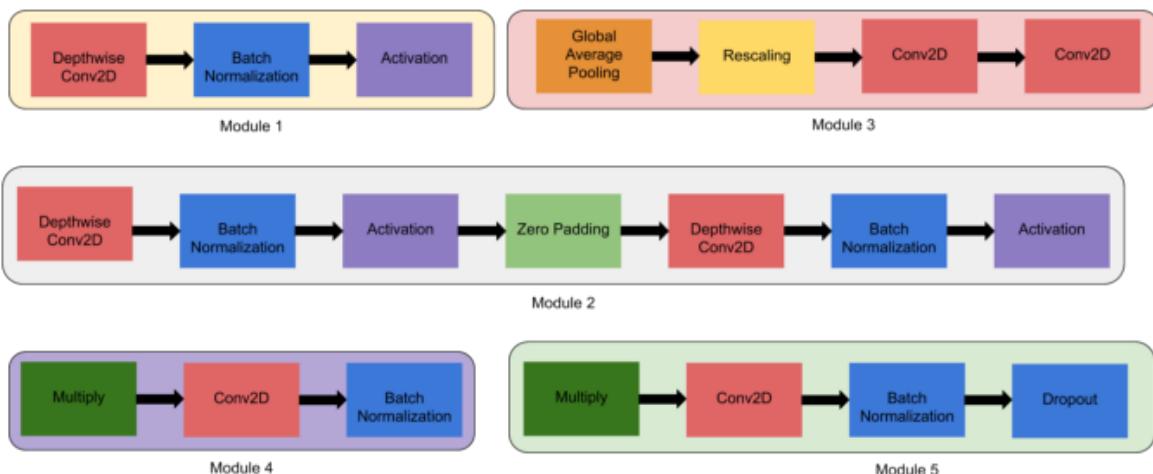


Figura 2.2: Architettura EfficientNet

Per *scaling* intendiamo la capacità di una rete neurale convoluzionale di essere *ampliata* in profondità (*depth*), larghezza (*width*), e nei coefficienti di risoluzione (*resolution coefficients*) dei vari layer.

Reti più *profonde* (numero di layer convoluzionali maggiore) possono catturare features più complesse ed in numero maggiore a scapito della maggiore difficoltà di training dovuta al problema del *vanishing gradient* [4].

Reti più *large* (lavorano con tensori di dimensione maggiore) tendono a essere più facilmente allenabili e a catturare piccoli dettagli.

Reti che utilizzano immagini a risoluzioni maggiori tendono a catturare dei pattern più complessi e raffinati.

EfficientDet-Lite 0, rispetto a EfficientDet-Lite 4, è meno *larga*, meno *profonda* e lavora con immagini in ingresso a risoluzione più bassa (320x320 vs

640x640) e permette perciò di ottenere i risultati delle detection in tempi più rapidi a discapito della precisione.

EfficientDet-Lite 4, invece, ha una latenza maggiore ma è in grado di fornire risultati più accurati in situazioni più complesse.

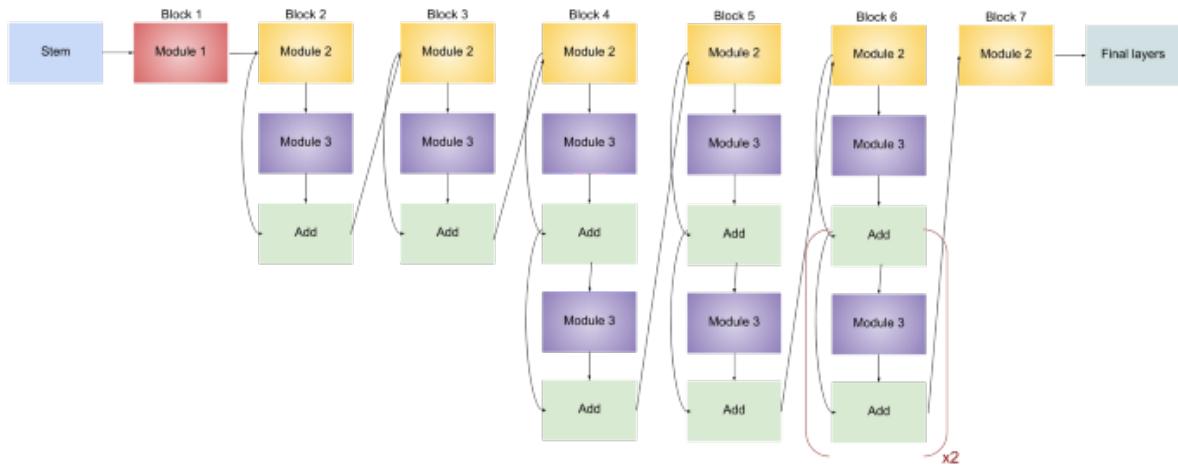


Figura 2.3: Architettura EfficientDet-Lite 0

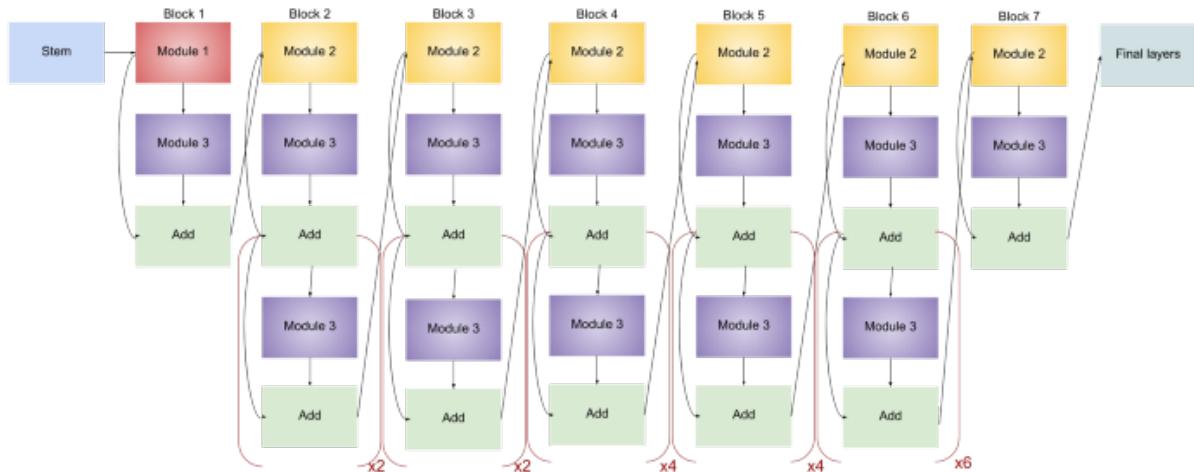


Figura 2.4: Architettura EfficientDet-Lite 4

Architettura	Dimensioni(MB)¹	Latenza²	Precisione Media³
EfficientDet-Lite 0	4.4	37	25.69%
EfficientDet-Lite 1	5.8	49	30.55%
EfficientDet-Lite 2	7.2	69	33.97%
EfficientDet-Lite 3	11.4	116	37.70%
EfficientDet-Lite 4	19.9	260	41.96%

Tabella 2.1: Confronto tra i modelli di EfficientDet-Lite

1. Dimensioni dei modelli dopo la quantizzazione intera.
2. Latenza misurata usando un Pixel 4 e la CPU con 4 thread.
3. La mAP (*mean Average Precision*) valutata sul COCO validation dataset del 2017.

2.4 Training e risultati

Per l'addestramento delle reti neurali abbiamo suddiviso il dataset in:

- 80% - Training
- 10% - Validation
- 10% - Testing

Le nostre reti basate su EfficientDet-Lite 0 e EfficientDet-Lite 4 sono state allenate per 50 epochhe e hanno raggiunto una precisione media rispettivamente del 61% e del 66%.

Di seguito i grafici delle loss di bounding-box e classi:

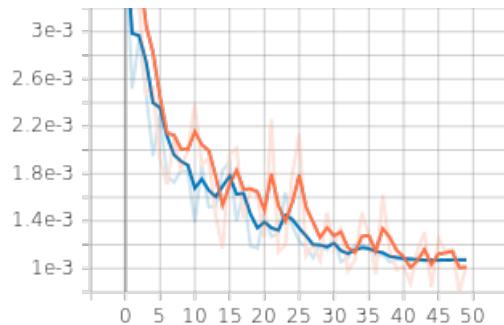


Figura 2.5: Box Loss - EfficientDet-lite 0

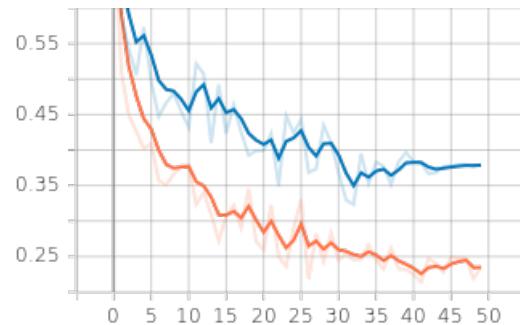


Figura 2.6: Class loss - EfficientDet-lite 0

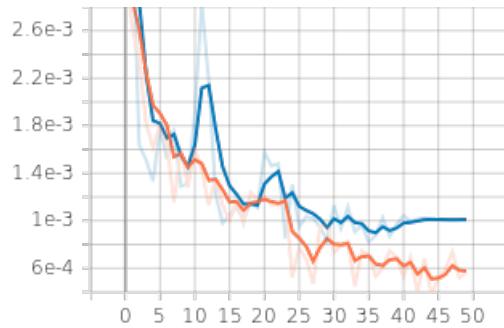


Figura 2.7: Box loss - EfficientDet-lite 4 (*pre-augmentation*)

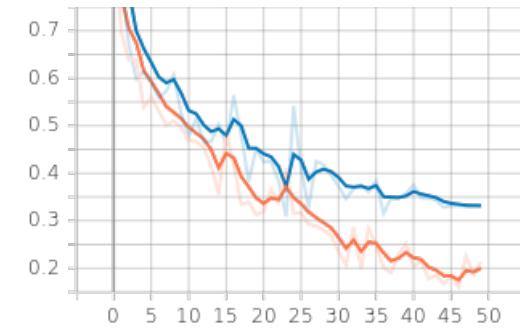


Figura 2.8: Class loss - EfficientDet-lite 4 (*pre-augmentation*)

Osservando il grafico della class-loss dell'EfficientDet-lite 4 abbiamo notato il fenomeno dell'*overfitting* [5] ed abbiamo perciò deciso di allenare nuovamente la rete utilizzando un pool di immagini migliorato.

Per il perfezionamento del dataset abbiamo deciso di usare le tecniche di data augmentation presentate successivamente (*Capitolo 4.2*).

A seguito dell'augmentation, la rete neurale ha raggiunto una precisione media dell'81%, risultato che abbiamo ritenuto soddisfacente.

Di seguito i grafici dell'EfficientDet-lite 4 allenata con il dataset *augmentato*:

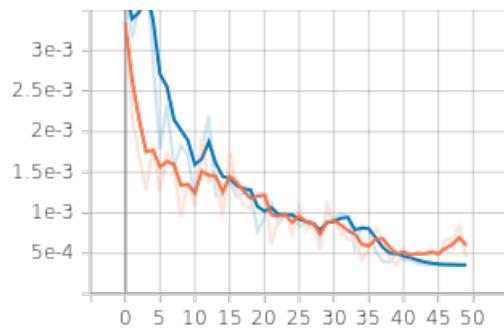


Figura 2.9: Box loss - EfficientDet-lite 4 (*post-augmentation*)

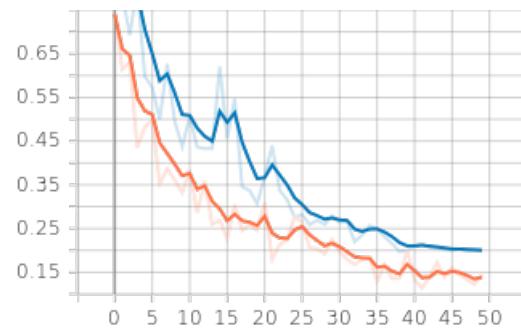


Figura 2.10: Class loss - EfficientDet-lite 4 (*post-augmentation*)

Capitolo 3

Architettura Applicazione

3.1 Descrizione generale

Al fine di realizzare le funzionalità precedentemente descritte, abbiamo creato due activity Android distinte: **MainActivity** ed **AnalyzeActivity**. La prima, appoggiandosi all'API CameraX, implementa le funzionalità necessarie alla *live detection* delle mascherine e alla cattura di immagini dalla fotocamera; la seconda, sfruttando l'API MediaStore, permette di salvare gli scatti nella galleria del telefono ed analizzarli in maniera più accurata rispetto alla live detection; entrambe si basano sul modello applicativo a callback.

3.2 MainActivity

MainActivity si occupa della gestione dei permessi che consentono l'accesso a fotocamera e allo storage del dispositivo.

Al momento della sua creazione, effettua inoltre una richiesta asincrona per la creazione di un'istanza della rete neurale EfficientDet-lite 0 ed inizializza le componenti legate all'acquisizione di frame tramite la classe **CameraProvider** (che a sua volta utilizza CameraX).

```
1 public class MainActivity extends AppCompatActivity {
2     private final String[] permissions = {
3         Manifest.permission.CAMERA,
4         Manifest.permission.READ_EXTERNAL_STORAGE,
5         Manifest.permission.WRITE_EXTERNAL_STORAGE
6     };
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         //...
10        //Request all the permissions needed if not already available
11        if(!permission.checkSelfPermission(this, permissions)) {
12            permission.requestPermission(this, permissions);
13        }
14    }
15 }
//...
16 public class Permission extends AppCompatActivity {
17     final static int PERMISSION_REQUEST_CODE = 100; //Arbitrary code
18
19     public boolean checkPermission(Context context, String[] permissions) {
20         for(String permission : permissions) {
21             if(ContextCompat.checkSelfPermission(context, permission)
22                 != PackageManager.PERMISSION_GRANTED) {
23                 return false;
24             }
25         }
26         return true;
27     }
28
29     public void requestPermission(Activity activity, String[] permissions) {
30         ActivityCompat.requestPermissions(
31             activity,
32             permissions,
33             PERMISSION_REQUEST_CODE);
34     }
35 }
36
37 }
```

Figura 3.1: Richiesta permessi applicazione

3.3 CameraProvider

CameraProvider è la classe responsabile della gestione delle funzionalità legate alla fotocamera; in particolare utilizza due *usecase* di CameraX chiamati *ImageCapture* e *ImageAnalysis* che permettono all’utente di catturare il frame corrente e di analizzare quelli acquisiti istante per istante.

Questa view si occupa inoltre di configurare l’hardware della fotocamera del dispositivo in modo da ottimizzare le performance in fase di ottenimento delle immagini.

```
1 private void startCamera(int lensOrientation) {
2     //...
3     this.provider = ProcessCameraProvider.getInstance(this.context);
4     this.provider.addListener(() -> {
5         try {
6             ProcessCameraProvider cameraProvider = this.provider.get();
7             cameraProvider.unbindAll(); //Clear use cases
8
9             //Image Capture
10            this.imageCapt = new ImageCapture.Builder()
11                .setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY)
12                .setTargetRotation(Surface.ROTATION_0).build();
13
14            //Image Analysis
15            this.imageAnalysis = new ImageAnalysis.Builder()
16                .setBackpressureStrategy(
17                    ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
18                .setOutputImageRotationEnabled(true)
19                .setOutputImageFormat(
20                    ImageAnalysis.OUTPUT_IMAGE_FORMAT_YUV_420_888)
21                .build();
22            this.imageAnalysis.setAnalyzer(this.analyzeExecutor, this::analyze);
23
24            UseCaseGroup useCaseGroup = new UseCaseGroup.Builder()
25                .setViewPort(this.previewView.getViewPort())
26                .addUseCase(this.preview)
27                .addUseCase(this.imageCapt)
28                .addUseCase(this.imageAnalysis).build();
29            this.camera = cameraProvider
30                .bindToLifecycle(this.context, cameraSelector, useCaseGroup);
31                //...
32        }, this.context.getMainExecutor());
33    }
```

Figura 3.2: Setup hardware della fotocamera tramite API CameraX

Per quel che riguarda l'acquisizione di una singola immagine abbiamo realizzato un'apposita funzione *captureImage* che si affida al metodo *takePicture* di CameraX per ottenere il frame corrente, con l'angolazione corretta (approfondimento a *Capitolo 4.1*), e sfrutta il suo callback per convertire l'immagine in formato *Bitmap* e creare un'istanza dell'AnalyzeActivity a cui passare, in modo asincrono, il frame.

```

1 public void captureImages() {
2     if(!this.isCameraAvailable) return;
3
4     //Take the picture
5     this.imageCapt.takePicture(
6         this.imageCaptureExecutors,
7         new ImageCapture.OnImageCapturedCallback() {
8             @Override
9             public void onCaptureSuccess(@NonNull ImageProxy imageProxy) {
10                 Bitmap frame = imageUtility.convertImageToBitmap(
11                     imageProxy.getImage(),
12                     getRotationDegree(imageProxy),
13                     currentLensOrientation == CameraSelector.LENS_FACING_FRONT
14                 );
15                 imageProxy.close();
16
17                 //Open a new analyze activity which will handle any error
18                 EventBus.getDefault().postSticky(new ClearSelectedDetectionEvent());
19                 EventBus.getDefault().postSticky(new PictureTakenEvent(frame));
20                 context.startActivity(new Intent(context, AnalyzeActivity.class));
21             }
22             /**
23             */
24         );
25     }

```

Figura 3.3: Metodo di acquisizione frame

La funzionalità di *live detection* dei volti è invece realizzata tramite lo usecase *ImageAnalysis* e si limita ad invocare, se richiesto dall'utente, il metodo di detection della rete neurale EfficientDet-Lite 0. I risultati vengono poi resi disponibili, in modo asincrono, alle view che si occupano della rappresentazione dei vari artefatti grafici, ovvero quelle responsabili di mostrare a schermo la posizione e i dettagli delle detection tramite rettangoli e pop-up informativi.

```
1 public void analyze(ImageProxy imageProxy) {
2     if (!this.liveDetection) {
3         imageProxy.close();
4         return;
5     }
6
7     //Load the image into a tensor
8     final TensorImage tensorImage = new TensorImage();
9     tensorImage.load(imageProxy.getImage());
10
11    //...
12
13    //Rotate and scale the image aquired from the sensor
14    final float translateX = this.flipNeeded ?
15                    (analyzeImageWidth - screenWidth) * 0.5:
16                    (screenWidth - analyzeImageWidth) * 0.5;
17    final float translateY = 0.5F * (screenHeight - analyzeImageHeight);
18    float scaleX = originalImageResolution.getWidth() / screenWidth;
19    float scaleY = originalImageResolution.getHeight() / screenHeight;
20    final float scalingFactor = scaleX > scaleY ?
21                    screenHeight / analyzeImageHeight :
22                    screenWidth / analyzeImageWidth;
23
24    //previewView's scale type is FILL_CENTER, so
25    //the transformations have the center of the previewView as the pivot
26    Matrix matrix = new Matrix();
27    matrix.preTranslate(translateX, translateY);
28    matrix.postScale(
29        scalingFactor, scalingFactor, 0.5F * screenWidth, 0.5F * screenHeight);
30
31    List<Detection> detections =
32        CameraProviderView.objectDetector.detect(tensorImage);
33
34    EventBus.getDefault().post(new UpdateDetectionsRectsEvent(...));
35    imageProxy.close();
36 }
```

Figura 3.4: Metodo che realizza la funzionalità di live detection

3.4 AnalyzeActivity

AnalyzeActivity si occupa di mostrare l'immagine catturata/caricata dall'utente e di fornire gli strumenti per salvarla sul dispositivo ed effettuare un'analisi più precisa rispetto alla funzionalità di *live detection* descritta precedentemente.

Al momento della sua creazione, effettua una richiesta asincrona per la creazione di un'istanza della rete neurale EfficientDet-lite 4, il cui metodo di detection viene invocato al momento della pressione del bottone *Analyze*.

```
1 public class AnalyzeActivity extends AppCompatActivity {
2     protected void onCreate(Bundle savedInstanceState) {
3         //...
4         //Asynchronously load the neural network
5         new Thread(() -> {
6             if(objectDetector == null) {
7                 objectDetector = new CustomObjectDetector(
8                     this,
9                     CustomObjectDetectorType.HIGH_ACCURACY
10                );
11            }
12        }).start();
13    }
14 }
```

Figura 3.5: Richiesta di creazione di un'istanza di EfficientDet-Lite 4

```

1 public class CustomObjectDetector {
2     public CustomObjectDetector(Context context, CustomObjectDetectorType type) {
3         final Builder defaultOptionsBuilder = ObjectDetectorOptions.builder()
4             .setBaseOptions(BaseOptions.builder().setNumThreads(4).build());
5         final Builder unsupportedOptionsBuilder = ObjectDetectorOptions.builder()
6             .setBaseOptions(BaseOptions.builder().useNnapi().build());
7
8         //Distinguish between HIGH ACCURACY and HIGH PERFORMANCE neural networks
9         try {
10             if (type == CustomObjectDetectorType.HIGH_ACCURACY)
11                 this.detector = ObjectDetector
12                     .createFromFileAndOptions(
13                         context,
14                         MODEL_FILE_F16,
15                         defaultOptionsBuilder.setMaxResults(1).build()
16                     );
17             else
18                 this.detector = ObjectDetector
19                     .createFromFileAndOptions(
20                         context,
21                         MODEL_FILE_I08,
22                         defaultOptionsBuilder.setMaxResults(10).build()
23                     );
24         } catch (Exception e) {
25             try {
26                 if (type == CustomObjectDetectorType.HIGH_ACCURACY)
27                     this.detector = ObjectDetector
28                         .createFromFileAndOptions(
29                             context,
30                             MODEL_FILE_F16,
31                             unsupportedOptionsBuilder.setMaxResults(1)
32                                 .build()
33                         );
34             else
35                 this.detector = ObjectDetector
36                     .createFromFileAndOptions(
37                         context,
38                         MODEL_FILE_I08,
39                         unsupportedOptionsBuilder.setMaxResults(10)
40                             .build()
41                         );
42         } catch (Exception e2) {...}
43     }
44 }

```

Figura 3.6: Creazione dell’istanza *generica* di EfficientDet-Lite

La funzionalità di salvataggio delle foto sul dispositivo viene realizzata tramite l'utilizzo delle nuove API MediaStore disponibili da Android 10+. Rispetto ai metodi d'interfacciamento con lo storage del dispositivo utilizzati nelle versioni precedenti di Android che davano libero accesso al filesystem sottostante, queste API forniscono un'astrazione tale da poter immaginare lo storage come una mappa chiave-valore pubblica in cui inserire le proprietà dei media che si stanno utilizzando, ottenendo un URI tramite cui è possibile salvare il contenuto del file (accesso al filesystem da parte dell'applicazione limitato tramite tecnica di *sandbox* [6]).

```

1  private ImageSavedEvent saveImage(Bitmap image) {
2      //Es. SISDIG_2021127_189230.jpg
3      final String pictureName = "SISDIG_" + new SimpleDateFormat("yyyyMMdd_HHmmss")
4          .format(new Date()) + ".jpeg";
5      //Create the picture's metadata (<Key, value> Map)
6      ContentValues newPictureDetails = new ContentValues();
7      newPictureDetails.put(MediaStore.Images.Media._ID, pictureName);
8      newPictureDetails.put(MediaStore.Images.Media.DISPLAY_NAME, pictureName);
9      newPictureDetails.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
10     newPictureDetails.put(MediaStore.Images.Media.WIDTH, image.getWidth());
11     newPictureDetails.put(MediaStore.Images.Media.HEIGHT, image.getHeight());
12     newPictureDetails.put(MediaStore.Images.Media.RELATIVE_PATH,
13             Environment.DIRECTORY_DCIM + "/SistemiDigitaliM");
14
15     //Add picture to MediaStore in order to make it accessible to other apps
16     //The result of the insert is the handle to the picture inside the MediaStore
17     Uri picturePublicUri = this.context.getContentResolver().insert(
18         Images.Media.EXTERNAL_CONTENT_URI, newPictureDetails);
19
20     //Saves the image and post the result on the EventBus
21     try {
22         OutputStream stream = this.context.getContentResolver()
23             .openOutputStream(picturePublicUri);
24         boolean imageSavedCorrectly = image.compress(CompressFormat.JPEG, 95, stream);
25         if (!stream.close() || !imageSavedCorrectly) throw new Exception(...);
26         return new ImageSavedEvent("success", picturePublicUri);
27     } catch (Exception e) {
28         //Remove the allocated space in the MediaStore if the picture can't be saved
29         this.context.getContentResolver().delete(picturePublicUri, new Bundle());
30         return new ImageSavedEvent(e.getMessage(), picturePublicUri);
31     }
32 }
```

Figura 3.7: Salvataggio delle immagini su filesystem tramite API MediaStore

Capitolo 4

Problemi affrontati

4.1 Rotazione fisica della fotocamera del dispositivo

Ogni smartphone disponibile in commercio dal 2010 ad oggi è equipaggiato con almeno una fotocamera frontale ed una fotocamera posteriore.

L'orientamento fisico con cui queste fotocamere possono essere montate sul dispositivo non è però standard e varia da telefono a telefono: utilizzando CameraX per catturare uno o più frame si nota immediatamente come i risultati ottenuti siano inconsistenti.

Dato che le reti neurali da noi utilizzate hanno bisogno di foto in input con tipo di orientamento costante (*portrait-orientation*), abbiamo realizzato un piccolo algoritmo per ruotare correttamente ogni foto scattata da ogni singolo dispositivo compatibile con CameraX.

```
1 private int getRotationDegree(ImageProxy imageProxy) {
2     int rotationDirection = currentLensOrientation == LENS_FACING_BACK ? 1 : -1;
3     int constantRotation = imageProxy.getImageInfo().getRotationDegrees() -
4         this.camera.getCameraInfo().getSensorRotationDegrees();
5
6     return this.camera.getCameraInfo().getSensorRotationDegrees() -
7         this.currentDisplayRotation +
8         constantRotation * rotationDirection;
9 }
```

Figura 4.1: Algoritmo per ottenere la corretta rotazione dei frame acquisiti tramite CameraX

4.2 Data augmentation

Per tentare di risolvere il problema dell’overfitting nell’allenamento della EfficientDet-lite 4 abbiamo deciso di *augmentare* il pool delle immagini utilizzate.

Per effettuare l’augmentation abbiamo creato un tool ad hoc scritto in python3 (disponibile a questo [link github](#)) che, oltre ad effettuare l’augmentation delle immagini, procede anche alla modifica dei file YOLO che contengono le rispettive *label* effettuando tutti i cambi di coordinate necessari. Di seguito, i tipi di data augmentation da noi utilizzati:

- **Blurring** (sfocatura dell’immagine con relativa perdita di dettagli)
- **Contrast** (aumento del contrasto tra i colori dell’immagine che porta in evidenza le *macrozone* principali)
- **Brightness** (aumento del fattore di luminosità dell’immagine, con attenuazione delle differenze tra le *macrozone* principali)
- **Grayscale** (messa in scala di grigi dell’immagine, con perdita totale di dettagli dati dal colore)
- **Flip** (distorsione dell’immagine, ottenuta specchiando l’immagine originale lungo uno o più assi)
- **Shear** (distorsione dell’immagine, ottenuta tramite stretching dell’immagine originale lungo uno o più assi)
- **Noise** (aggiunta di rumore gaussiano all’interno dell’immagine con relativa perdita di *micro-dettagli*)
- **Rotation** (rotazione rispetto al centro dell’immagine di un angolo casuale)
- **Zoom** (distorsione dell’immagine, effettuata tramite uno zoom sulla parte centrale)
- **Sharpening** (aumento della nitidezza dell’immagine, mettendo in rilievo i *micro-dettagli*)
- **Translation** (traslazione dell’immagine di un numero casuale di pixel, eventualmente tralasciando parti dell’immagine)

Per velocizzare il processo di augmentation abbiamo utilizzato la libreria **CuPy**, la quale mette a disposizione delle versioni di **NumPy** e **SciPy** che impiegano la **GPU** per la computazione.

```
1  try: #test for full-gpu support
2      import cupy as np
3      from cupyx.scipy import ndimage
4      from cupyx.scipy.ndimage.filters import gaussian_filter
5  except:
6      import numpy as np
7      from scipy import ndimage
8
9  class AugmenterGrayscale(Augmenter):
10     def __init__(self):
11         self.__transformation_matrix__ = np.ndarray([1, 0, 0, 0],
12                                         [0, 1, 0, 0],
13                                         [0, 0, 0, 0],
14                                         [0, 0, 0, 1]])
15
16     #Applies the convolution kernel
17     def transform(self, image: Image):
18         return ndimage.affine_transform(input = self.get_array_from_image(image),
19                                         matrix = self.__transformation_matrix__)
20
21     def get_transformed_YOLO_values(self, center_x, center_y, width, height):
22         return center_x, center_y, width, height
23
24     def get_augmenter_signature(self):
25         return "GRSCL"
```

Figura 4.2: Esempio di una classe utilizzata per l'augmentation del dataset

Capitolo 5

Risultati ottenuti

I risultati ottenuti sono relativamente coerenti con ciò che ci aspettavamo dopo la fase di allenamento delle reti neurali.

Abbiamo però riscontrato che introducendo variabili non presenti in fase di training, i risultati possono variare in maniera più o meno considerevole. Alcuni di questi cambiamenti possono essere rappresentati da indumenti particolari o semplici monili indossati dai soggetti.

Nelle pagine seguenti, sono riportati alcuni esempi di rilevazioni corrette ed incorrette.

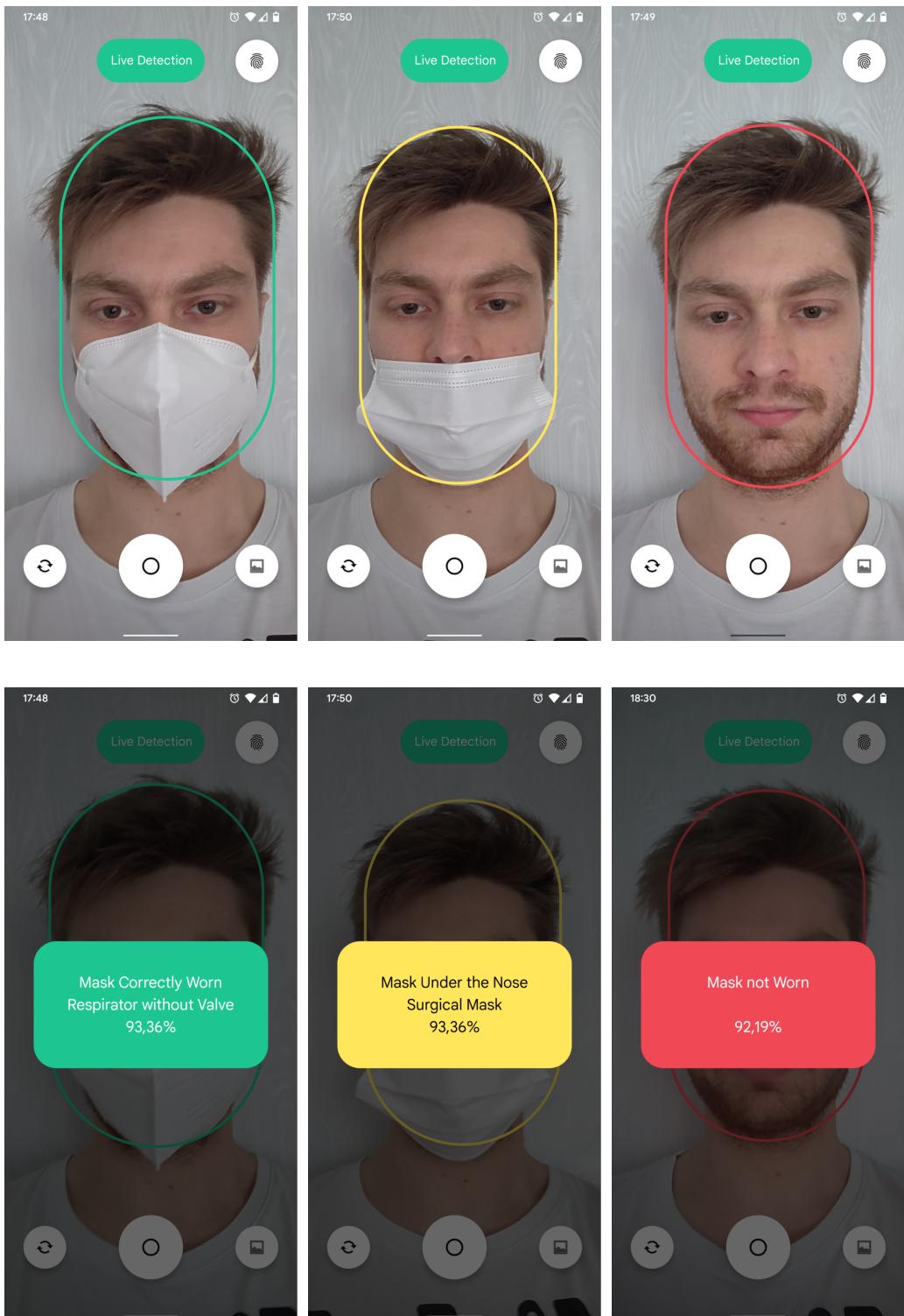


Figura 5.1: Esempi di rilevazioni correttamente effettuate

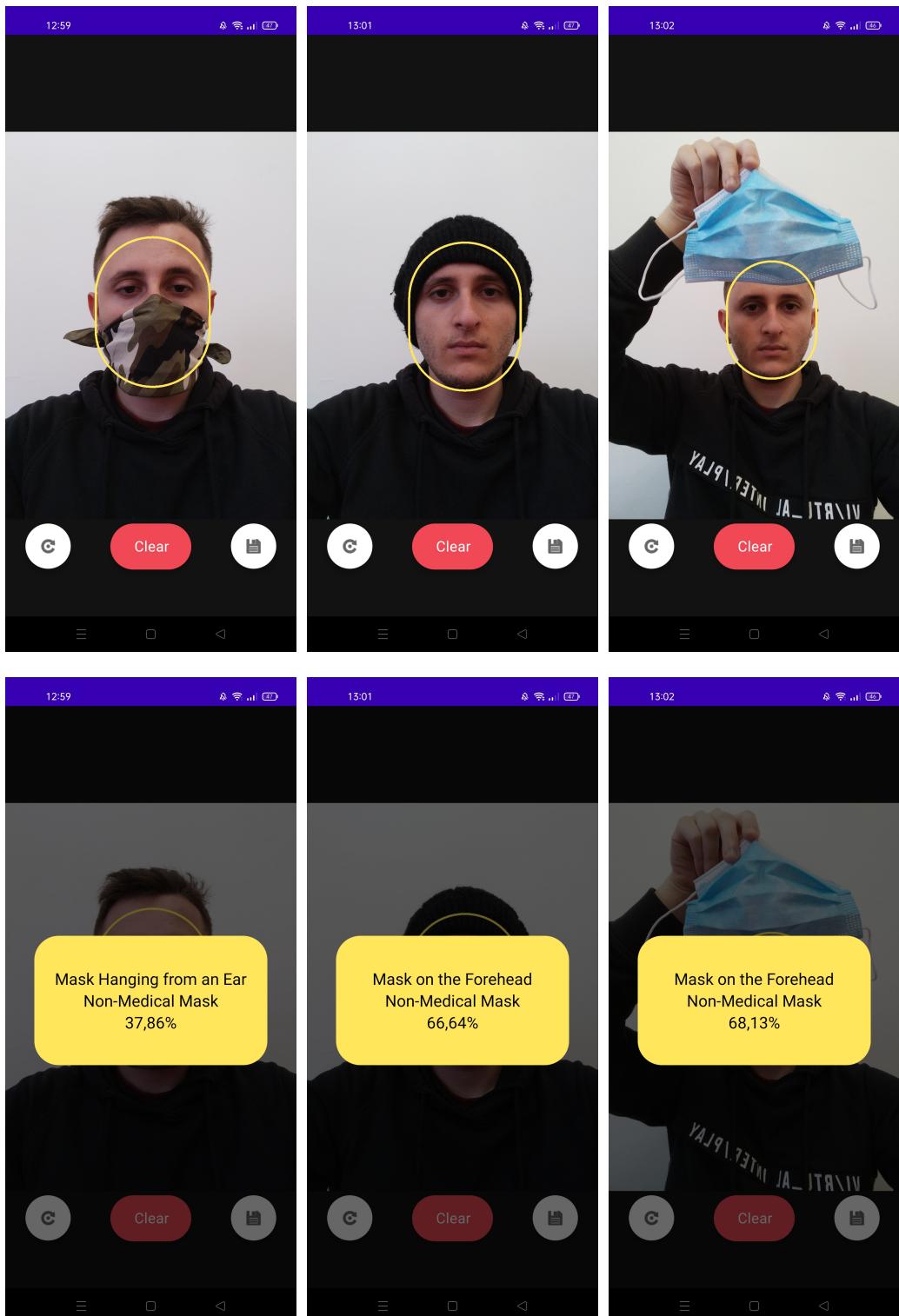


Figura 5.2: Esempi di rilevazioni effettuate incorrettamente

Capitolo 6

Estensione del progetto

6.1 Abstract

Sulla base di quanto realizzato nel progetto originale, abbiamo deciso di realizzare una funzionalità aggiuntiva che permetta di capire se due o più persone, presenti nelle immagini catturate dall'applicazione, stanno o meno rispettando la distanza di sicurezza di **1m** presente nelle normative anti-covid.

6.2 Metodo di risoluzione

L'idea alla base del nostro metodo è quella di partire da una singola immagine, acquisita tramite le funzionalità descritte in precedenza, ed utilizzare tecniche di monocular-depth estimation combinate ad un uso efficace di proporzioni e trigonometria, al fine di stimare le distanze tra soggetti.

Questo è possibile basandosi su una semplificazione fondamentale: tutte le distanze calcolate sono distanze relative tra volti umani, i quali hanno delle proporzioni ben precise che scalano a seconda della distanza tra loro e l'obiettivo.

In particolare, data un'immagine nella quale sono presenti almeno due persone (che chiameremo A e B) il primo passo è quello di utilizzare una rete neurale (nel nostro caso la rete MiDAS di Intel [7]) per ottenere una depth-map contenente le distanze relative tra pixel.

Tramite la funzionalità di detection della rete EfficientDet-Lite 4 realizzata in precedenza, si ottengono poi le coordinate dei rettangoli contenenti i volti di A e B e si sfruttano per calcolare la profondità media (indicata dalla depth-map) all'interno di queste due aree.

```

1  public float getAverageDepthInDetection(
2      float[] depthMap, float depthMapWidth, float depthMapHeight,
3      float left, float width, float top, float height) {
4          //Make sure to stay within the depthMap width/height constraints
5          left = left < 0 ? 0 : Math.min(left, depthMapWidth - 1);
6          top = top < 0 ? 0 : Math.min(top, depthMapHeight - 1);
7          final int availableWidth =
8              (int) Math.min(depthMapWidth - 1, left + width);
9          final int availableHeight =
10             (int) Math.min(depthMapHeight - 1, top + height);
11
12         float averageDepth = 0.0F;
13         for (int j = (int) top; j < availableHeight; j++) {
14             for (int i = (int) left; i < availableWidth; i++) {
15                 averageDepth += depthMap[(int)(i + j * depthMapWidth)];
16             }
17         }
18         return averageDepth / (availableWidth * availableHeight);
19     }

```

Figura 6.1: Calcolo della profondità media di un volto

A questo punto si sfruttano le dimensioni di un volto *standard* (266px x 353px con una foto di 1728px x 2304px) e quelle del rettangolo contenente il volto che risulta più lontano e si ottiene così la distanza tra l'osservatore e quella persona. Utilizzando le distanze date dalla depth-map si ottiene anche la distanza tra l'osservatore e la persona più vicina ad esso: rispetto ad utilizzare semplicemente le proporzioni tra volti, abbiamo riscontrato che questo procedimento riduce l'errore sulla misurazione finale della posizione lungo l'asse z.

$$f_{1_m} : 1_m = f_{d_i} : d_i \quad i \in [1..n_{faces}] \subset \mathbb{N} \quad (6.1)$$

da cui:

$$d_i = \frac{f_{1_m}}{f_{d_i}} \Rightarrow d_{max} = \frac{f_{1_m}}{f_{d_{max}}} \quad (6.2)$$

$$\varphi_{max} : d_{max} = \varphi_i : d_i \quad (6.3)$$

da cui:

$$d_i = \frac{d_{max} * \varphi_{d_{max}}}{\varphi_i} \quad (6.4)$$

f_{1_m} = dimensione in px del volto *standard* osservato ad 1_m di distanza

f_{d_i} = dimensione in px di un volto *generico* osservato ad una distanza d_i

φ_i = valore medio della depth map nell'area in cui è inscritto un volto osservato ad una distanza d_i

Le distanze tra A e B vengono combinate con le distanze (in px) tra i rettangoli contenenti i rispettivi volti ed il centro dello schermo (si forza il punto di fuga nel centro dell'immagine). Viene poi calcolata la distorsione, data dalla profondità, lungo gli assi x e y.

$$x_i = d_i * \mu_{x_i} * \Gamma \quad (6.5)$$

$$y_i = d_i * \mu_{y_i} * \Gamma \quad (6.6)$$

Piano Π = piano avente:

- l'asse x passante per la metà orizzontale dello schermo del telefono
- l'asse y passante per la metà verticale dello schermo del telefono
- l'asse z passante per il centro dello schermo (dall'osservatore all'orizzonte)

x_i = coordinata x del volto i su Π

y_i = coordinata y del volto i su Π

μ_{x_i} = distanza minima (in px) tra il volto i e l'asse orizzontale del piano Π

μ_{y_i} = distanza minima (in px) tra il volto i e l'asse verticale del piano Π

Γ = fattore di conversione px \Rightarrow m

Infine, per ottenere le coordinate lungo l'asse z, si tiene conto del fatto che A e/o B possano avere un'altezza diversa da quella dell'osservatore e che quindi la foto possa essere stata scattata con un'inclinazione diversa da 0° .

$$d_{i\pi} = \sqrt{\frac{d_i^2}{\Phi_i^2} - y_i^2}, \quad \Phi_i = \frac{Res_s}{Res_i} \quad (6.7)$$

$$z_i = \sqrt{d_{i\pi}^2 - x_i^2} \quad (6.8)$$

$d_{i\pi}$ = distanza d_i corretta tenendo conto dell'inclinazione originale della foto

Φ_i = rapporto tra la risoluzione della foto con cui sono state prese le misure *standard* e quella della foto attuale

z_i = coordinata z del volto i su Π

Avendo ora le coordinate lungo tutti e tre gli assi, aggiustate a seconda della distanza delle persone e dell'angolo a cui la foto è stata scattata, è sufficiente utilizzare la formula della distanza tra due punti per ottenere la distanza relativa tra A e B.

$$d_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2} \quad (6.9)$$

6.3 Problemi del metodo e risultati

Il metodo sopra descritto presenta alcune criticità.

In particolare, basandosi su misure di un volto *standard*, non si tengono in considerazione quelle variazioni di dimensioni che sono però presenti nella realtà.

Ad esempio il volto di un bambino può non rispettare completamente le proporzioni sopra indicate e/o variare di molto in dimensioni rispetto a quelle utilizzate per la misurazione delle coordinate sull'asse z.

Inoltre, in media, i volti femminili hanno dimensioni leggermente inferiori a quelli maschili (rappresentati dalle misure *standard*) ed anche in questo caso le misurazioni delle distanze dall'osservatore potrebbero risultare lievemente alterate.

In aggiunta, abbiamo riscontrato che la rete neurale MiDas non è accurata al 100% ed in alcuni casi, l'errata stima della profondità porta la depth-map a contenere valori inconsistenti e perciò ad influenzare negativamente le misurazioni finali delle distanze.

Infine, questo metodo non tiene conto della diversa lunghezza focale che possono avere le fotocamere dei dispositivi con cui è possibile acquisire le immagini: diverse lunghezze focali portano ad avere diverse distorsioni dell'immagine, alterando perciò la dimensione (in px) dei rettangoli contenenti i volti delle persone e di conseguenza le loro coordinate sull'asse z.

Di seguito vengono riportati alcuni dei risultati ottenuti.

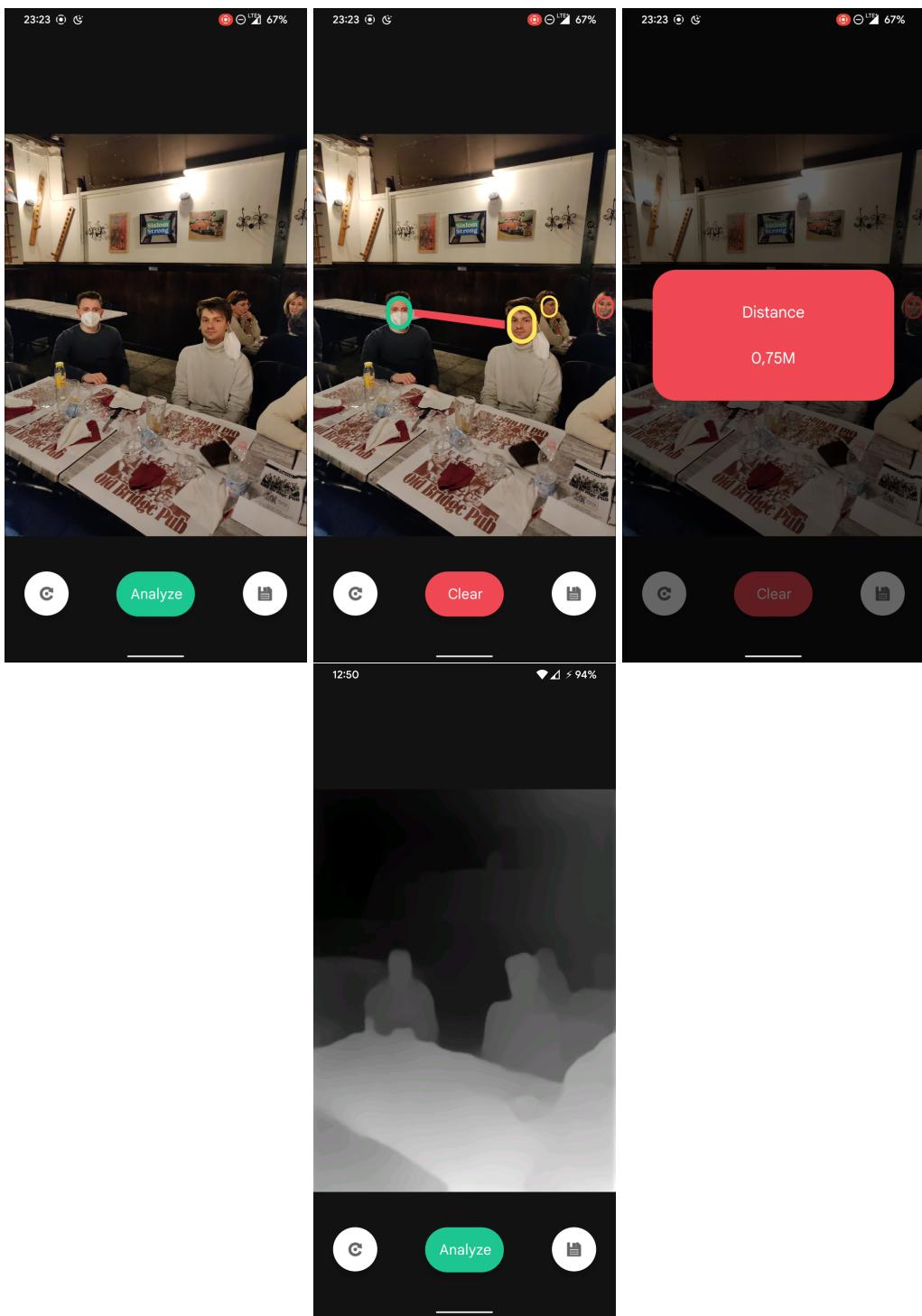


Figura 6.2: Esempio di distanza calcolata correttamente (misura reale: 70cm)

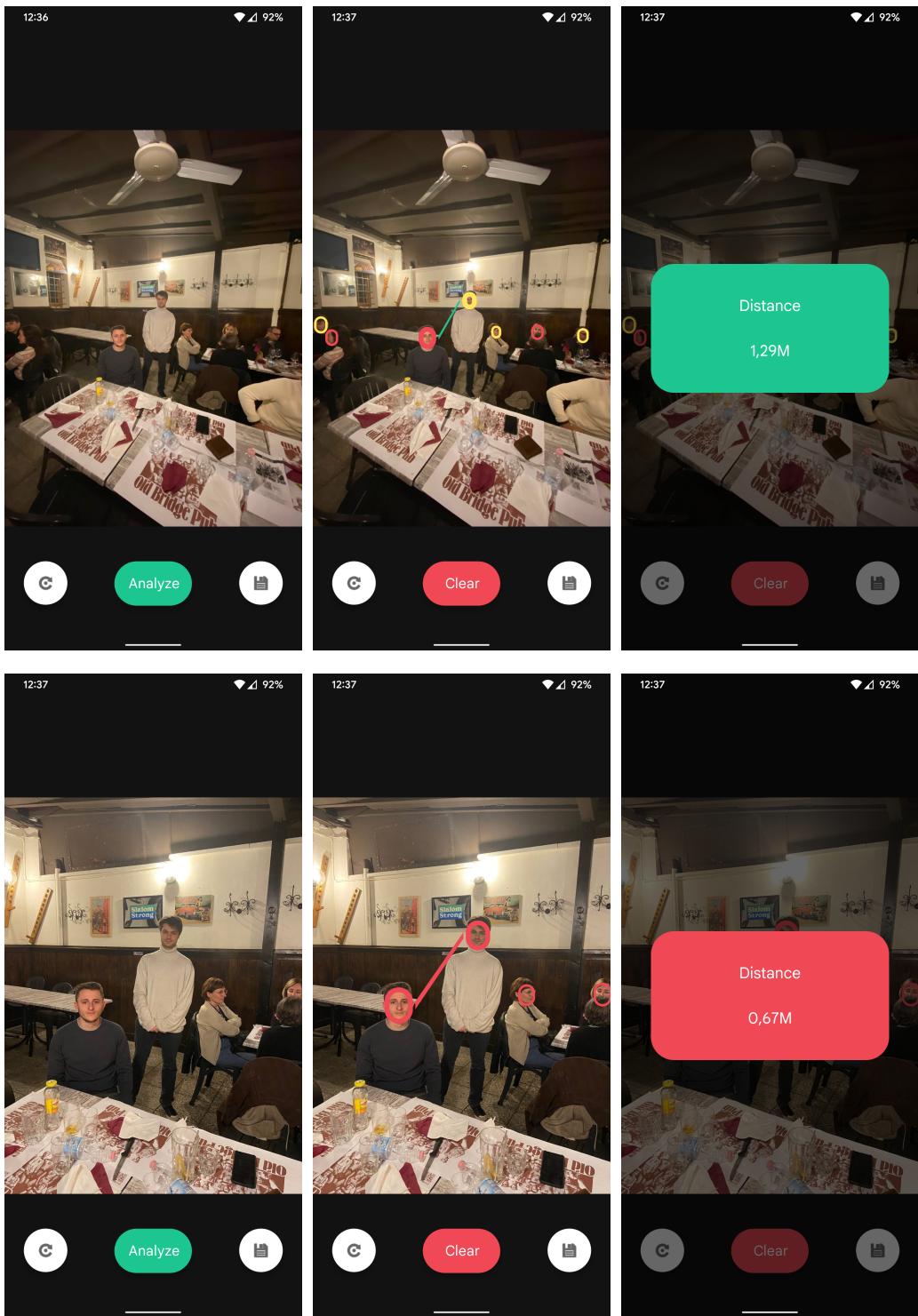


Figura 6.3: Esempio di errore introdotto dalla variazione della lunghezza focale della lente della fotocamera (misura reale: 1,32m)

Bibliografia

- [1] Bartolomeo Montrucchio Antonio Marceddu Renato Ferrero. *Ways to wear a mask or respirator(WWMR-DB)*. Apr. 2021. URL: <https://ieee-dataport.org/open-access/ways-wear-mask-or-respirator-wwmr-db>.
- [2] Mingxing Tan, Ruoming Pang e Quoc Le. *EfficientDet: Scalable and Efficient Object Detection*. Giu. 2020. URL: <https://arxiv.org/pdf/1911.09070.pdf>.
- [3] Mingxing Tan e Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2019. DOI: 10.48550/ARXIV.1905.11946. URL: <https://arxiv.org/pdf/1905.11946v5.pdf>.
- [4] Sepp Hochreiter. *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions*. Apr. 1998. DOI: 10.1142/S0218488598000094.
- [5] *The Overfitting phenomenon*. URL: <https://en.wikipedia.org/wiki/Overfitting>.
- [6] *Android 10 - Application Sandbox*. URL: <https://source.android.com/security/app-sandbox>.
- [7] Intel. *Rete Neurale MiDAS - Monocular Depth Estimation*. URL: <https://github.com/isl-org/MiDaS>.