
Applicazione per rilevare e oscurare volti e testi

# INDICE

# 1. Introduzione al progetto

Il progetto HIDE nasce con l'intento di creare un'applicazione Android che automaticamente riesca a rilevare e oscurare i volti e le scritte all'interno di immagini e video. Nello specifico, prevediamo tre modalità di utilizzo:

1) **Live:** il rilevamento e l'oscuramento avvengono in tempo reale, analizzando costantemente il flusso di immagini provenienti dalla fotocamera.
2) **Foto:** il rilevamento e l'oscuramento avvengono su un'immagine caricata dalla galleria o su una foto appena scattata.
3) **Video:** il rilevamento e l'oscuramento avvengono su un video appena registrato o caricato dalla galleria.

Per eseguire il rilevamento automatico realizzeremo e utilizzeremo dei modelli TensorFlow Lite in grado di riconoscere due classi di oggetti:

1) **Faccia:** riconoscere il volto di una persona, persino da angolazioni diverse.
2) **Testo:** riconoscere scritte differenti per grandezza, font e orientamento.

In conclusione, il nostro obiettivo è implementare un'applicazione che sia funzionante e veloce, ma allo stesso tempo anche esteticamente gradevole, funzionale e user-friendly.

# 2. Dataset

## 2.1. Selezione immagini

La prima fase del progetto consiste nella creazione del dataset, ovvero dell'insieme di immagini che saranno utilizzate per addestrare il modello da includere nell'applicazione Android.
Sfortunatamente, non esiste già un dataset che raccoglie ed etichetta insieme volti e testi; per questo abbiamo dovuto assemblarlo noi. Infatti, il nostro dataset nasce dall'unione di due noti dataset:

- **WIDER FACE:** dataset di riferimento per il rilevamento dei volti. È composto da 32.203 immagini in cui sono stati etichettati ben 393.703 volti con un alto grado di variabilità in scala, posa e occlusione ambientale. Tutte le immagini sono distribuite con un carico di 40% per train, 10% per validate e 50% per test [1].
- **COCO-Text:** dataset specializzato nel rilevamento del testo. Infatti, è formato da 63.686 immagini contenenti 239.506 scritte. COCO-Text prende le immagini dal noto dataset COCO, utilizzato per il rilevamento e la segmentazione di una vasta gamma di oggetti [2].

Unendo i due dataset soprastanti, siamo riusciti a ottenere un totale di ben 95.889 immagini. Purtroppo, non abbiamo potuto sfruttare tutte queste immagini perché la fase di addestramento sarebbe durata troppo, superando i limiti di utilizzo delle GPU imposto da Google Colab. A causa di ciò, siamo stati costretti a creare una versione più contenuta del nostro dataset di partenza.
Il nuovo dataset ottenuto è stato organizzato nel seguente modo:

1) **Training set (80%):** 20.000 immagini usate per addestrare il modello di rilevamento oggetti.
2) **Validation set (20%):** 5.000 immagini appartenenti allo stesso insieme di distribuzione del training set. Sono importanti per effettuare il fine-tuning degli iperparametri della rete e valutare se il modello riesce a generalizzare bene analizzando immagini che non ha mai visto prima.

In conclusione, il dataset iniziale è stato ridotto di circa il 75%, ottenendo un dataset finale di sole 25.000 immagini. Chiaramente, questo ridimensionamento porta con sé dei pro e dei contro.
Lo svantaggio principale è quello di usare meno immagini nella fase di training. Questo comporta la creazione di un modello meno accurato nella risoluzione generale del problema che, nel nostro caso, si traduce in una minore precisione nel riconoscimento di volti e testi. D'altro canto, esistono vari metodi per compensare questa problematica, come la tecnica del data augmentation che applica dei cambiamenti casuali ma controllati alle immagini fornite, ampliando di fatto il dataset di partenza. Inoltre, avere un dataset non molto grande è vantaggioso dal punto di vista delle tempistiche dell'addestramento, che si riducono sensibilmente.

## 2.2. Etichettatura immagini

Naturalmente, le immagini raccolte vanno etichettate. Per fortuna, i dataset WIDER FACE e COCO-Text contengono già le immagini con le relative etichette. L'unico problema è che non utilizzano lo stesso formato. Infatti, in WIDER FACE a ogni immagine si associa un file di testo in cui vengono salvate le coordinate del punto in alto a sinistra e le dimensioni (ampiezza e altezza) della box che delinea un certo oggetto rilevato nella scena. Invece, in COCO-Text si ha un unico file JSON che racchiude sia le informazioni di tutte le immagini sia le coordinate delle varie box.

Per il nostro progetto abbiamo deciso di utilizzare un terzo formato: Pascal VOC [3]. In quest'ultimo, a ogni immagine si associa un file XML che, oltre a contenere le informazioni sull'immagine stessa, contiene persino i dati delle varie box. Ad ognuna di esse si associano le coordinate del punto in alto a sinistra e di quello in basso a destra, necessari per disegnare la box sull'immagine.
La conversione dai formati di partenza al Pascal VOC non è stata semplice perché non esistono tool gratuiti che offrono la possibilità di effettuare questo tipo di conversione. Pertanto, abbiamo implementato in Python il nostro tool di conversione personalizzato che autonomamente analizza tutte le etichette delle immagini nei dataset WIDER FACE e COCO-Text e le converte nel formato Pascal VOC [4].
La scelta di questo formato è dipesa principalmente da tre motivi:

1) **Controllo con LabelImg:** questo programma ci permette di visualizzare agevolmente le immagini del nostro dataset per controllare se le etichette delle classi, faccia e testo, erano nella giusta posizione e avevano la dicitura corretta. In caso di problemi, è possibile effettuare delle modifiche velocemente. Da questo tipo di controllo abbiamo appurato che i dataset scelti erano qualitativamente adatti per il nostro scopo, sia per il gran numero di etichette presenti sia per la loro correttezza.
2) **Supporto TensorFlow Model Maker:** la libreria TensorFlow Model Maker mette nativamente a disposizione dei metodi per caricare e lavorare su dataset etichettati con il formato Pascal VOC.
3) **Coordinate box OpenCV:** la libreria OpenCV, alla stessa maniera di Pascal VOC, usa le coordinate del punto in alto a sinistra e di quello in basso a destra per disegnare la box sull'immagine. Di conseguenza, abbiamo compatibilità tra il modo in cui il modello TensorFlow Lite delinea le box e quello in cui OpenCV le andrà a disegnare sull'immagine.

Dopo la fase di etichettatura, il nostro dataset finale era pronto per essere utilizzato. Per importarlo più agevolmente su Google Colab e averlo sempre a disposizione da remoto, abbiamo deciso di caricare il dataset su **Google Drive** [5].

# 3. TensorFlow

## 3.1. Google Colab

Per la creazione e l'addestramento del modello TensorFlow Lite abbiamo utilizzato **Google Colab**. Lo abbiamo scelto perché è uno strumento gratuito della suite di Google che consente di eseguire codice Python direttamente dal proprio browser utilizzando i notebook Jupyter.
Infatti, per addestrare il nostro modello, abbiamo scritto un **notebook Jupyter** contenente il codice necessario e i relativi commenti [6]. Inoltre, il notebook può essere gestito come un documento condiviso su Google Drive in maniera tale da consentire a eventuali membri del gruppo di poterlo modificare in qualsiasi momento.
Le macchine virtuali messe a disposizione da Google Colab non solo sono in grado di eseguire codice Python, ma mettono a disposizione persino delle librerie utili per il deep learning come Keras e TensorFlow. Oltre a ciò, si può usufruire di GPU e TPU per dare un boost computazionale all'implementazione di reti neurali. La nostra macchina virtuale impiegava delle GPU Nvidia Tesla T4 (16 GB), 32 GB di RAM e un disco fisso da 256 GB.
Sicuramente l'hardware messo a disposizione dalla macchina virtuale di Google Colab ci ha permesso di eseguire l'addestramento in tempi relativamente brevi. L'unico svantaggio era legato al fatto che durante tutta la fase di addestramento abbiamo dovuto lasciare i nostri computer accesi per essere costantemente collegati con il notebook in esecuzione e, nonostante ciò, diverse volte la sessione di collegamento scadeva e dovevamo ricominciare tutto daccapo.

## 3.2. TensorFlow Lite Model Maker

Nel nostro progetto abbiamo usato la **libreria TensorFlow Lite Model Maker**, la quale semplifica il processo di addestramento di un modello TensorFlow Lite creato a partire da un dataset personalizzato.
La libreria Model Maker attualmente supporta le seguenti attività di Machine Learning:

- Classificazione immagini: classifica le immagini in categorie predefinite.
- Rilevamento oggetti: rileva oggetti in tempo reale.
- Classificazione del testo: classifica il testo in categorie predefinite.
- BERT domanda-risposta: trova la risposta in un determinato contesto per una determinata domanda con BERT.
- Classificazione audio: classifica l'audio in categorie predefinite.
- Raccomandazione: consiglia elementi in base alle informazioni di contesto per lo scenario sul dispositivo.
- Ricercatore: cerca testo o immagine simile in un database.

Naturalmente, da questo ampio insieme di attività disponibili, per il nostro scopo abbiamo scelto il rilevamento oggetti.
Al posto della libreria Model Maker avremmo potuto usare Keras: libreria di alto livello scritta in Python, utile per interfacciarsi con TensorFlow durante l'implementazione di reti neurali. Tra le due abbiamo preferito Model Maker per una questione di semplicità e immediatezza (l'immagine sottostante ne è la prova).

Without Model Maker

```
IMAGE_SIZE = 224
BATCH_SIZE = 64
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
   rescale=1./255,
   validation_split=0.2)
train_data = datagen.flow_from_directory('flower_photos/',target_size=(IMAGE_SIZE, IMAGE_SIZE), batch_size=BATCH_SIZE,subset='training')
test_data = datagen.flow_from_directory(base_dir, target_size=(IMAGE_SIZE, IMAGE_SIZE),batch_size=BATCH_SIZE,
subset='testing')

# Download the headless model
feature_extractor_url = "https://tfhub.dev/google/efficientnet/b0/feature-vector/1"
feature_extractor_layer = hub.KerasLayer(feature_extractor_url, input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
feature_extractor_layer.trainable = False

# Attach a classification head
model = tf.keras.Sequential([feature_extractor_layer, layers.Dense(train_data.num_classes, activation='softmax')])

# train the model
model.compile(optimizer=tf.keras.optimizers.Adam(),loss='categorical_crossentropy',metrics=['acc'])
steps_per_epoch = np.ceil(train_data.samples/train_data.batch_size)
history = model.fit(train_data, epochs=2,steps_per_epoch=steps_per_epoch)
class_names = sorted(test_data.class_indices.items(),key=lambda pair:pair[1])
class_names = np.array([key.title() for key, value in class_names])

# evaluate the model
for image_batch, label_batch in test_data:
  predicted_batch = model.predict(image_batch)
  predicted_id = np.argmax(predicted_batch, axis=-1)
  predicted_label_batch = class_names[predicted_id]
model.evaluate(test_data)

# export the model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with open('flower.tflite', 'w') as f:
  f.write(tflite_model)
```

With Model Maker

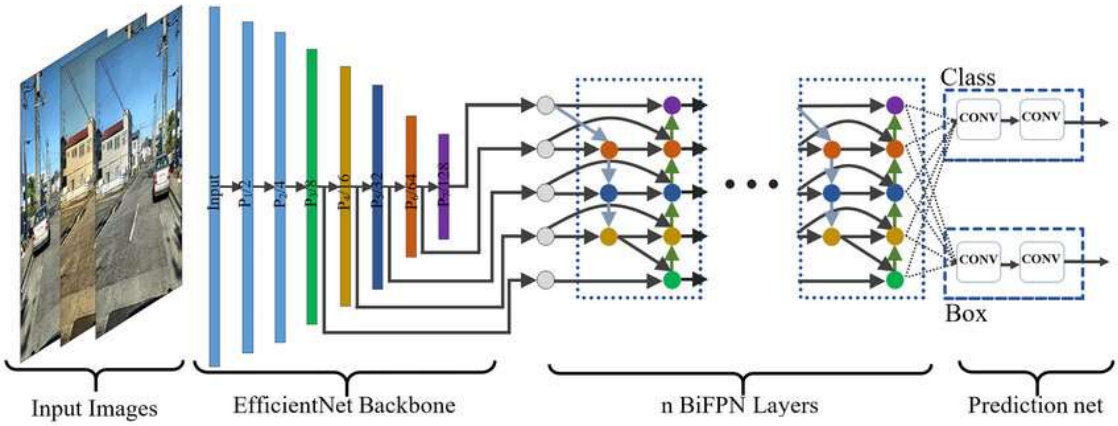
```
# 1. Load data.
train_data, val_data, test_data =
   ObjectDetectorDataloader.from_csv('dataset.csv')


# 2. Customize the model.
model = image_classifier.create(train_data)


# 3. Evaluate the model.
loss, accuracy = model.evaluate(test_data)


# 4. Export to tflite.
model.export('android_detector.tflite')
```

## 3.3. Addestramento modello di rilevamento oggetti TensorFlow Lite

### 3.3.1. Scelta architettura del modello

Come già visto nei paragrafi precedenti, tutto il codice per effettuare l'addestramento è inserito all'interno di un notebook Jupyter ed è eseguito su una macchina virtuale di Google Colab.
Ora analizziamo più nel dettaglio il codice del notebook. Innanzitutto, sulla macchina virtuale devono essere importati sia i pacchetti software necessari per l'addestramento sia il dataset da Google Drive. Dopo aver estratto l'archivio del dataset e aver suddiviso le immagini ottenute in base al set di appartenenza, bisogna selezionare l'architettura del modello. La scelta di quest'ultima ricade tra i modelli di rilevamento oggetti di tipo mobile-friendly appartenenti alla famiglia EfficientDet-Lite.
L'architettura EfficientDet nasce con l'intento di offrire un modello di rilevamento oggetti scalabile ed efficiente [7]. Per far ciò, si avvale principalmente due ottimizzazioni:

1) Una rete pesata a piramide BiFPN (Bidirectional Feature Pyramid Network) che implementa una fusione di funzionalità multilivello facile e veloce, consentendo alle informazioni di fluire in entrambe le direzioni (dall'alto verso il basso e viceversa) attraverso delle connessioni regolari ed efficienti. La rete BiFPN aggiunge un ulteriore peso a ciascuna funzione di input per discriminarle in base alla loro rilevanza.



2) Un metodo di ridimensionamento composto che ridimensiona in modo uniforme la risoluzione, la profondità e la larghezza per tutte le reti backbone, feature network e box/class forecast contemporaneamente.

A partire dall'architettura EfficentDet è stata derivata la EfficientDet-Lite, perfezionata per il mondo del mobile e dell'IoT. Analizzando la tabella delle prestazioni di ciascun modello EfficientDet-Lite, abbiamo deciso di addestrare ben due modelli per la nostra applicazione:

- **EfficientDet-Lite0:** modello dalle dimensioni ridotte, molto veloce ma con una precisione media bassa. Ideale per la modalità live della nostra applicazione, dove è necessario elaborare velocemente il flusso delle immagini proveniente dalla fotocamera, senza preoccuparsi troppo della precisione [8].
- **EfficientDet-Lite2:** modello che supera il precedente sia per precisione media sia per latenza. Ideale per la modalità foto e video, in cui la velocità di elaborazione passa in secondo piano a favore di una maggiore precisione nel rilevare volti e testi per ottenere risultati di qualità superiore [9].

Una volta scelta l'architettura del modello, si passa all'addestramento vero e proprio.

### 3.3.2. Addestramento del modello TensorFlow

L'addestramento del modello TensorFlow avviene grazie alla classe ObjectDetector che sfrutta le immagini del training set per addestrarsi e quelle del validation set per effettuare il fine-tuning degli iperparametri della rete.

Sulla base delle nostre esigenze e dell'hardware messo a disposizione dalla macchina virtuale di Google Colab, per il nostro addestramento abbiamo scelto le seguenti opzioni:

- **epoch = 30:** durante l'addestramento il training set sarà esaminato per 30 volte. È importante stare attenti all'accuratezza della validation e fermarsi quando il valore di val_loss smette di diminuire per evitare overfitting.
- **batch_size = 20:** saranno necessari 1.000 step per esaminare tutte le 20.000 immagini del training set.
- **train_whole_model = True:** agevola il fine-tuning dell'intero modello, evitando di allenare solamente il layer head per migliorare l'accuratezza. Purtroppo, questa opzione dilata i tempi di addestramento.

Inoltre, siccome stiamo utilizzando la libreria Model Maker, durante l'addestramento viene sfruttato anche il **transfer learning**: partendo dalle reti di rilevamento oggetti con architettura EfficientDet, già preaddestrate sul dataset ImageNet, noi le andiamo a riconvertire e specializzare per il rilevamento di volti e testi. Così facendo, riusciamo a realizzare un modello di rete neurale che sfrutta persino tutti i vantaggi di efficienza e scalabilità dell'architettura EfficientDet.

Non è stata usata la **data augmentation**, ovvero la generazione di nuove immagini a partire dalle immagini di partenza a cui si applicano dei cambiamenti (rotazioni, capovolgimenti, tagli, modifiche del colore) casuali ma controllati, perché il dataset conteneva già un ampio set di immagini diverse.

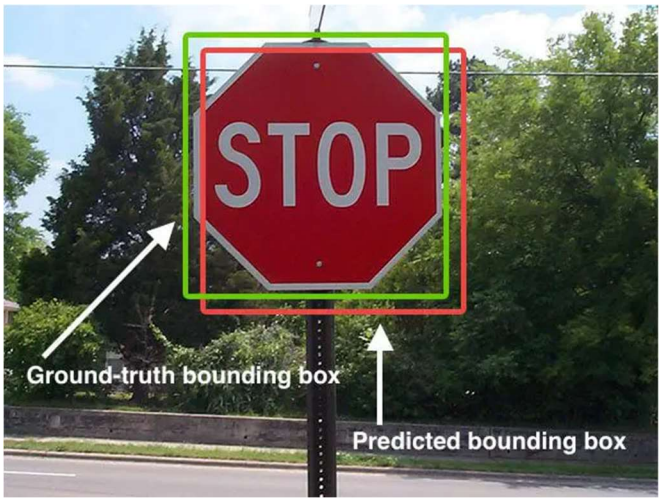
### 3.3.3. Esportazione come modello TensorFlow Lite

Il modello ottenuto dall'addestramento è nel formato TensorFlow ed è troppo oneroso da elaborare per un'applicazione Android che esegue su dispositivi mobile, non dotati di hardware molto

performanti. Per questo motivo, il modello TensorFlow deve essere esportato in un formato più idoneo, ovvero TensorFlow Lite.
Un modello TensorFlow Lite, identificato dall'estensione .tflite, è rappresentato in un formato efficiente e facilmente trasportabile noto come FlatBuffers. Tra i suoi principali vantaggi ci sono le dimensioni ridotte e l'inferenza più veloce; possibile grazie al fatto che l'accesso ai dati avviene senza ulteriori operazioni di parsing o di deserializzazione.
Durante la fase di esportazione, il modello viene compresso tramite delle tecniche di quantizzazione, ovvero riducendo la precisione dei valori utilizzati per rappresentare i parametri del modello che di default sono espressi in Floating-point a 32 bit. Con la quantizzazione il modello guadagna velocità e riduce i suoi consumi, ma subisce anche un peggioramento dell'accuratezza che scende di circa 1% rispetto al modello originale. Nel nostro caso, abbiamo usato una quantizzazione full-integer che ha convertito il modello da FP32 a INT8.
Il modello di rilevamento oggetti TensorFlow Lite ottenuto dopo l'esportazione è pronto per essere scaricato e inserito all'interno dell'applicazione Android.

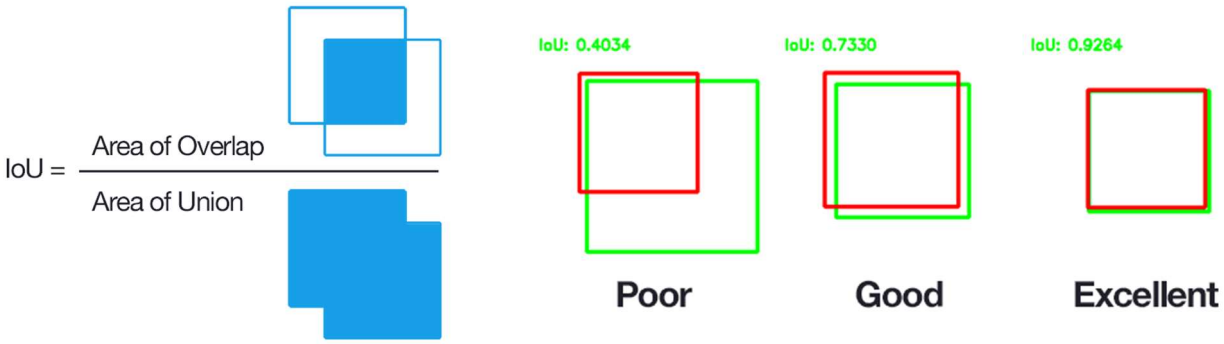
## 3.4. Considerazioni sui modelli

Una volta ottenuti i due modelli TensorFlow Lite, possiamo valutarne l'accuratezza. Come metodo di valutazione abbiamo scelto IoU (Intersection Over Union), ovvero il valore che quantifica il grado di sovrapposizione tra due box. La prima rappresenta l'area in cui si trova effettivamente l'oggetto (Ground-truth); la seconda identifica l'area in cui è stato rilevato lo stesso oggetto (Prediction) [10].




Il valore della IoU può variare da 0 a 1, dove 0 significa che la predizione della posizione è completamente sbagliata; mentre 1 che la predizione della posizione è stata perfetta. Chiaramente, nella realtà è praticamente impossibile ottenere una IoU pari a 1 perché il modello che utilizziamo subisce diverse manipolazioni e approssimazioni che rendono impossibile un rilevamento senza imprecisioni. Sapendo ciò, la metrica IoU è stata pensata come il rapporto tra l'area di intersezione tra le due box e l'area di unione delle stesse box. Questo permette alla IoU di valutare come migliori le previsioni in cui si ha una maggiore sovrapposizione tra le due box. In questo tipo di metrica la cosa importante non è che le coordinate delle box coincidano fra di loro ma che l'area di Prediction sia il più possibile simile a quella di Ground-truth.
Sicuramente, maggiore è il valore della IoU e più il modello è capace di individuare con accuratezza la posizione dell'oggetto rilevato nello spazio. In generale, un modello di rilevamento oggetti ha una

scarsa precisione se la sua IoU è al di sotto del 65%, buona precisione tra 65% e 85%, eccellente precisione sopra 85%.




Per quanto riguarda i nostri modelli, loro hanno una valutazione è molto buona visto che è pari a 73% per EfficientDet-Lite0 e a 76% per EfficientDet-Lite2.
Per calcolare la IoU dei nostri modelli, non siamo riusciti a trovare dei tool già pronti quindi ci siamo costruiti i nostri [11]. Il primo tool è scritto in Java e serve per analizzare tutte le foto del validation set, ottenendo tutti i dati associati alle box degli oggetti presenti in queste immagini. Per il rilevamento, si sfrutta lo stesso codice utilizzato anche dall'applicazione. Il secondo tool, invece, è scritto in Python e serve per calcolare il valore di IoU per i due modelli. In questo tool si confrontano le box Ground-truth, i cui dati sono presi dai file di annotazione presenti nel validation set, con le relative box Prediction, i cui dati sono stati calcolati dal primo tool. L'algoritmo del secondo tool calcola prima la IoU per ogni immagine e poi ne fa la media per ottenere la IoU del modello stesso.

# 4. Android

## 4.1. Implementazione applicazione Android

Una volta ottenuto il modello di rilevamento oggetti TensorFlow Lite, si passa all'implementazione dell'applicazione Android [12]. Come ambiente di sviluppo abbiamo scelto **Android Studio** perché ci è sembrato il più completo e semplice da utilizzare tra tutti quelli disponibili. Per codificare l'applicazione abbiamo scelto **Java**, linguaggio con cui abbiamo più familiarità e che conosciamo maggiormente rispetto a Kotlin. La grafica dell'applicazione è stata interamente realizzata da noi sfruttando il tool grafico basato su layout XML e integrato in Android Studio.
L'applicazione utilizza anche alcune librerie esterne, che ci permettono di gestire più semplicemente diverse operazioni complesse:

- **CameraX:** libreria Jetpack creata per semplificare lo sviluppo di app in cui si utilizza la fotocamera. Fornisce un'API facile da utilizzare e supportata dalla maggior parte dei dispositivi Android (5.0 in poi).
- **OpenCV:** libreria open source per la visione artificiale e l'apprendimento automatico. Nella nostra app è stata impiegata principalmente per oscurare i volti e i testi, modificando l'immagine di partenza.
- **TensorFlow Lite Task:** libreria che offre delle interfacce per relazionarsi con il modello TensorFlow Lite integrato nell'app. Le interfacce sono ottimizzate in base al tipo di attività che il modello deve eseguire in maniera tale da ottenere il massimo delle prestazioni da quest'ultimo.
- **JCodec:** libreria che implementa le codifiche audio e video più popolari. Ci torna utile nella modalità video quando dobbiamo assemblare il nuovo video contenente tutti i frame con i volti e i testi oscurati.

Per realizzare tutte le funzionalità viste nel primo capitolo, abbiamo deciso di strutturare l'applicazione in due activity: Start Activity e Main Activity.

## 4.2. Start Activity

La Start Activity è stata concepita sia come schermata iniziale, utile per accogliere l'utente non appena apre l'applicazione, sia come controllore dei permessi. Infatti, affinché l'applicazione funzioni completamente e correttamente, l'utente deve concedere i seguenti permessi:

- **Camera:** permesso di poter usare la fotocamera.
- **Record audio:** permesso di registrare audio durante i video.
- **Write external storage:** permesso per scrivere file sulla memoria interna ed esterna.
- **Read external storage:** permesso di leggere file dalla memoria interna ed esterna.

Nello specifico, la Start Activity è stata programmata in maniera tale che l'utente non possa passare alla Main Activity fino a quando non concede tutti i permessi richiesti. Questo meccanismo è possibile notarlo nelle seguenti immagini: sulla sinistra la schermata in cui l'utente ha concesso tutti i permessi richiesti e può entrare nella Main Activity; sulla destra la schermata in cui si richiede all'utente di concedere i permessi mancanti.



## 4.3. Main Activity

La Main Activity è la schermata principale in cui l'utente ha la possibilità di utilizzare l'applicazione nelle tre modalità previste (live, foto e video) e perfino di modificare le sue impostazioni [13].

### 4.3.1. Modalità live

La **modalità live** è pensata per il rilevamento e l'oscuramento di visi e testi in tempo reale, analizzando costantemente il flusso di immagini provenienti dalla fotocamera. Quest'ultimo è ottenuto grazie alla libreria CameraX che mette a disposizione una particolare view chiamata PreviewView, la quale è in grado di mostrare a schermo tutto ciò che la fotocamera cattura. Un altro componente importante è Image Analyzer, che incorpora la funzione di analisi del flusso di immagini provenienti dalla fotocamera.
Catturare, analizzare e modificare le immagini per oscurare volti e testi è ovviamente un compito oneroso e farlo in tempo reale presenta diverse criticità. Innanzitutto, l'analisi e la modifica devono essere eseguite il più velocemente possibile per evitare stuttering. Inoltre, queste operazioni sono eseguite in maniera continuativa, quindi, potrebbero bloccare la GUI dell'applicazione. Per risolvere questi problemi, abbiamo deciso di delegare la parte di analisi e di modifica delle immagini a un thread separato rispetto a quello principale. Così facendo, non solo evitiamo che la GUI si blocchi ma riusciamo persino a velocizzare e ottimizzare l'elaborazione delle immagini, riducendo sensibilmente il problema dello stuttering. Purtroppo, riusciamo a garantire un'alta fluidità solamente se il rilevamento di visi e testi viene fatto con il modello EfficientDet-Lite0. Con il modello EfficientDet-Lite2 l'elaborazione in tempo reale è più lenta perché questo modello sacrifica la velocità di rilevamento a favore di una maggiore precisione.

### 4.3.2. Modalità foto

La **modalità foto** è stata creata per il rilevamento e l'oscuramento di visi e testi su un'immagine caricata dalla galleria o su una foto appena scattata. Nel primo caso, basterà semplicemente cliccare sull'apposita icona per aprire la galleria e scegliere la foto desiderata. Nel secondo caso, la foto viene

scattata catturando l'immagine mostrata sulla PreviewView nell'istante in cui si preme l'icona dello scatto. Indipendentemente dal metodo utilizzato, l'immagine da analizzare viene salvata in una variabile globale come bitmap per essere facilmente identificabile durante l'analisi successiva.
L'elaborazione di una singola immagine non è un compito particolarmente complesso e oneroso. Difatti, per questa modalità non abbiamo previsto particolari ottimizzazioni e consigliamo di utilizzare il modello EfficientDet-Lite2 per avere una maggiore precisione nel rilevamento di volti e testi. Usando il modello EfficientDet-Lite0 non si ottiene nessun miglioramento particolare in termini di velocità di elaborazione dell'immagine.

### 4.3.3. Modalità video

La **modalità video** è nata per il rilevamento e l'oscuramento di visi e testi su un video appena registrato o caricato dalla galleria. Il caricamento del video dalla galleria funziona esattamente come il caricamento di un'immagine visto nella modalità foto. Invece, per quanto riguarda la registrazione di un video, il meccanismo è un po' più complesso. Quando si decide di registrare un video, si apre un flusso di output che cattura i frame proveniente dalla PreviewView e li salva nel file del video. Terminata la registrazione o la selezione dalla galleria, l'URI (Uniform Resource Identifier) del video viene salvato globalmente in maniera tale da poterlo identificare senza problemi nella fase di analisi.
L'analisi del video è un processo oneroso perché il rilevamento e l'oscuramento di visi e testi avviene frame per frame. Questo comporta elaborazioni molto lunghe a causa del gran numero di immagini da analizzare: basti pensare che in un singolo secondo di un video a 30 FPS sono contenute già 30 immagini. Se elaborassimo il video sul thread principale, la GUI dell'applicazione si bloccherebbe per tutto il tempo di elaborazione, rendendo l'app inutilizzabile.
Per evitare ciò e per ottimizzare l'elaborazione del video, anche in questo caso abbiamo pensato di creare un thread staccato da quello principale, in grado di effettuare il rilevamento e l'oscuramento senza causare blocchi all'applicazione. Più nello specifico, dal video registrato o caricato si estraggono i singoli frame, i quali vengono analizzati (rilevamento), modificati (oscuramento) e infine riassemblati in un nuovo video, grazie alla libreria JCodec. Il video appena elaborato viene salvato nella memoria interna dello smartphone per offrire all'utente la possibilità di poterlo vedere più volte, senza la necessità di rielaborare ogni volta lo stesso video di partenza.
In questa modalità, consigliamo di usare il modello EfficientDet-Lite0 se si vuole ottenere un'elaborazione più veloce e il modello EfficientDet-Lite2 per un'analisi più precisa.

### 4.3.4. Impostazioni

Le **impostazioni** giocano un ruolo fondamentale nella personalizzazione del rilevamento di volti e testi. Non solo è possibile scegliere quale classe di oggetti rilevare, ma anche qual è la soglia minima di confidenza che gli oggetti di una certa classe devono rispettare per esser poi effettivamente considerati nella successiva fase di oscuramento. Inoltre, è possibile stabilire quanti oggetti devono essere rilevati. Purtroppo, il limite massimo è 25 ed è imposto da TensorFlow Lite.
Infine, è possibile scegliere il tipo di modello da utilizzare: per elaborazioni più veloci (prestazioni) viene selezionato EfficientDet-Lite0; mentre per elaborazioni più precise (precisione) viene scelto EfficientDet-Lite2.


## 4.4. Rilevamento e oscuramento di volti e testi

Il rilevamento e l'oscuramento di volti e testi avviene sempre su un'immagine:

- Modalità live: l'immagine proviene dal flusso di immagini della fotocamera.
- Modalità foto: l'immagine viene scattata con la fotocamera o caricata dalla galleria.
- Modalità video: l'immagine è il frame del video che si sta elaborando.

Il **rilevamento** lo gestiamo mediante la libreria TensorFlow Lite Task. Quest'ultima ci mette a disposizione una serie di funzioni utili per poter utilizzare il nostro modello TensorFlow Lite e individuare i volti e i testi all'interno di un'immagine. Innanzitutto, si parte analizzando il bitmap dell'immagine scelta. Questa analisi serve per rilevare gli oggetti nella scena, taggandoli con apposite etichette di categoria e specificando le coordinate della box che li contiene. Successivamente, ogni oggetto rilevato viene controllato e se il suo punteggio di confidenza supera la soglia impostata nelle impostazioni, allora quell'oggetto viene oscurato.

L'**oscuramento** viene gestito grazie alla libreria OpenCV, che si occupa di modificare l'immagine in maniera tale da oscurare i volti e i testi rilevati. Nello specifico, a ogni oggetto rilevato è collegata una box, ovvero un rettangolo che delimita l'area dell'immagine in cui esso è contenuto.

Per oscurare un oggetto seguiamo i seguenti passi:

1) Ritagliare la parte che contiene l'oggetto dall'immagine originale, ottenendo una nuova immagine.
2) L'immagine ritagliata subisce poi delle modifiche sottoforma di ridimensionamenti, in maniera tale da ottenere l'effetto di pixelizzazione.
3) Infine, l'immagine ritagliata e modificata viene riposizionata nella sua posizione iniziale, dando l'effetto di oscuramento dell'oggetto che si trovava nella stessa area dell'immagine iniziale.

Questo meccanismo viene iterato per tutti gli oggetti rilevati, ottenendo alla fine un'immagine in cui sono tutti oscurati.

# Riferimenti


1. Dataset WIDER FACE, http://shuoyang1213.me/WIDERFACE/
2. Dataset COCO, https://cocodataset.org/#download
3. Formato Pascal VOC, https://www.section.io/engineering-education/understanding-pascal-voc-dataset/#pascal-voc-taxonomy
4. Tool per la conversione da altri formati in Pascal VOC, https://github.com/francesco-paglia/Hide/blob/main/Tools/DatasetManager/main.py
5. Dataset finale, https://github.com/francesco-paglia/Hide/blob/main/Dataset/Link%20dataset.txt
6. Notebook Jupyter per l'addestramento del modello di rilevamento oggetti, https://github.com/francesco-paglia/Hide/blob/main/TensorFlow%20Lite/Addestramento%20Hide.ipynb
7. Architettura EfficientDet, https://paperswithcode.com/method/efficientdet
8. Modello TensorFlow Lite EfficientDet-Lite0, https://github.com/francesco-paglia/Hide/blob/main/TensorFlow%20Lite/Modelli/android0.tflite
9. Modello TensorFlow Lite EfficientDet-Lite2, https://github.com/francesco-paglia/Hide/blob/main/TensorFlow%20Lite/Modelli/android2.tflite
10. Spiegazione IoU, https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation
11. Tool per il calcolo del IoU, https://github.com/francesco-paglia/Hide/blob/main/Tools/IoU.java

    https://github.com/francesco-paglia/Hide/blob/main/Tools/IoU/main.py
12. Applicazione Android, https://github.com/francesco-paglia/Hide/tree/main/Android/Codice
13. Dimostrazione funzionamento applicazione, https://github.com/francesco-paglia/Hide/blob/main/Dimostrazione/Link%20video.txt