Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Implementazione di Driver Checker: un'app per la prevenzione dell'uso del telefono alla guida.

Relatore:
Prof. Federico Montori

Presentata da:
David Mack

Sessione
Anno Accademico 2022/2023

A mia madre e la mia famiglia,
che mi supporta e mi sopporta . . .

Introduzione

Il cellulare, o *smartphone*, per usare un termine più attuale, è uno strumento molto recente che ha completamente stravolto la società. L'idea è stata introdotta da Apple quando nel *9/01/2007* è stato presentato il primo iPhone dal fondatore *Steve Jobs*. In questi ultimi 16 anni sono cambiate molte cose rispetto alla prima versione, tant'è che è difficile compararlo in termini di prestazioni ed utilizzo. Oramai lo smartphone è diventato parte integrante di una persona, viene utilizzato in tutti gli ambiti della vita quotidiana, per navigare su internet, per mantenere i contatti con famigliari e amici sparsi nel mondo, per guardare video sulle piattaforme come YouTube e per giocare ai videogiochi.

Il report più recente condotto dall'azienda Data Reportal e pubblicato a ottobre del 2023 [1], rappresenta molte statistiche sull'uso del cellulare da parte della popolazione mondiale. Uno di queste è il tempo medio d'utilizzo dello smartphone, come visibile nella figura 1, nella fascia d'età tra i 16 ed i 64 anni è stata registrata una media di **6:41** ore giornaliere nel penultimo trimestre del 2023. Come si può immaginare questo uso sproporzionato del cellulare ha delle conseguenze negative nella società, come discusso sul sito [4]: in alcuni casi può essere considerato una vera e propria dipendenza. Un'altra conseguenza negativa è l'aumento degli incidenti causati dall'uso dello smartphone. Questo ultimo fenomeno è stato definito come *texting and driving* e ci sono stati degli studi al riguardo. Infatti, i dati descritti dal rapporto annuale di Istat e Aci, pongono l'uso del cellulare alla guida come la causa principale di incidenti stradali con morti e feriti. Nel 2020, circa il 15.7% dei sinistri con lesioni registrati in Italia sono dovuti al texting and driving, causando 2.395 vittime e 159.249 persone con conseguenze fisiche più o meno gravi [6]. In questi ultimi anni per contrastare questa tendenza sono state

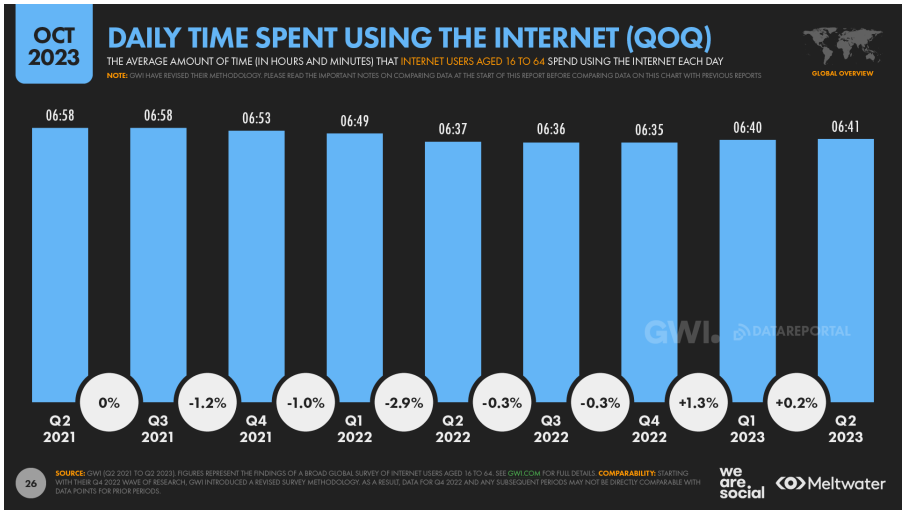


Figura 1: L'immagine rappresenta la quantità di ore medie nella popolazione mondiale tra 16 e 64 anni [3]

sviluppate delle applicazioni con l'intenzione di migliorare le proprie abitudini di guida. Per incentivare le persone ad utilizzare l'app normalmente ti permettono di guadagnare come ad esempio SmanApp che offre una compensazione economica in cambio del rispetto dei limiti di velocità oppure DriveCoin che allo stesso modo sensibilizza il seguire i limiti di velocità in cambio di sconti o offerte personalizzate da parte di aziende locali.
Dal punto di vista legislativo, l'uso del cellulare alla guida non è concesso se non tramite un dispositivo di **vivavoce** o con un **auricolare** a patto che chi si trova al volante ci senta bene dalle due orecchie. Se non si è in possesso del viva voce o dell'auricolare, l'automobilista ha due opzioni: o accosta e si ferma per fare la telefonata o per rispondere ad una chiamata oppure dimentica di avere con sé il cellulare e attende l'arrivo a destinazione. A questo proposito, va ricordato che il Codice della strada obbliga a tenere sempre entrambe le mani sul volante, tranne quando si deve cambiare la marcia. Quindi, già il fatto di tenere una mano occupata con il cellulare è motivo sufficiente per essere multati. Negli ultimi mesi sono state aggiornate le sanzioni riguardo l'uso del telefono alla guida con un inasprimento della sanzione pecuniaria, che passa da 165-660 euro a 422-1.697 euro, sospensione della patente dai 15 giorni ai 2 mesi e la decurtazione di 8 punti patente sin dalla prima violazione. In caso di recidiva nei successivi due anni, oltre alla sospensione della patente da 1 a 3 mesi, si prevede il pagamento di una somma da

644 a 2.588 euro e la decurtazione di 10 punti patente [5, 6, 7].

Le applicazioni di cui abbiamo parlato antecedentemente non sono delle vere e proprie soluzioni, infatti promuovono la persona a seguire i limiti di velocità e non al diminuire l'uso dello smartphone però lo stato italiano ha cercato di trovare un modo per contrastare questa piaga:

Telecamere anti-distrazioni Sono stati utilizzati due approcci differenti, il primo, quello tradizionale, che usa degli agenti di polizia negli incroci ed il metodo moderno con l'uso di telecamere. Il problema principale è che sono in precisi punti della strada, portando il guidatore a seguire le regole sono in tali luoghi.

Uso dei dati telematici L'uso dei dati prodotti dalle black box o l'uso dei sensori del cellulare, come il giroscopio, per rilevare quando il telefono viene utilizzato durante la guida. Ma c'è un problema principale, cioè che non è ancora possibile determinare se la persona che sta utilizzando lo smartphone sia il passeggero o il guidatore.

Tutte le soluzioni trovate però hanno un problema principale, non hanno la funzione di **prevenzione**.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'Arte

Il rilevamento dell'uso del telefono durante la guida è un ramo dell'Activity Recognition, la quale ha l'obiettivo di riconoscere le azioni e gli scopi di uno o più agenti a partire da una serie di osservazioni sulle azioni degli stessi e sulle condizioni ambientali.

Per riuscire nel nostro intento è necessario risolvere alcune incognite quali: riconoscere che ci sia un utilizzo attivo dello smartphone, che l'utilizzatore sia nella posizione del guidatore e che il conducente stia effettivamente guidando. In passato ci sono stati degli studi che hanno tentato di risolvere tali problematiche, tra questi troviamo [9] che aveva l'obiettivo di riconoscere in quale mezzo di trasporto si trovava la persona, utilizzando il giroscopio presente nel telefono. Un altro lavoro molto utile è quello descritto nella tesi di Octavian Bujor [10] il quale, sempre attraverso il giroscopio, comprende se la persona che utilizza il cellulare è sul lato destro o sinistro dell'automobile. Un'altra ricerca molto interessante fatta in questo campo riguarda lo studio dei movimenti della testa del guidatore durante la guida [12]. Lo studio utilizza dei dispositivi esterni per analizzare costantemente il guidatore. L'obiettivo della ricerca è riuscire a riconoscere quando il conducente è distratto dalla guida, causa del cellulare o causa della sonnolenza. I test eseguiti hanno dato dei risultati molto promettenti con circa l'84% di accuratezza. L'ultimo studio che andremo a presentare riguarda l'utilizzo dei sensori prensenti nel cellulare per riconoscere il suo utilizzo da parte del guidatore [13]. La ricerca si basa sulla idea che l'utilizzo del cellulare alla guida porta il guidatore a delle azioni inusuali rispetto al suo normale utilizzo. Infatti viene riconosciuto un rallentamento della velocità della

(a) Collezione dei dati da nella ricerca [12]

(b) Esempio di device esterno ADAS

Figura 1.1: Immagini relative a ricerche nell'ambito del *texting and driving*

macchina, una posizione leggermente differente del cellulare ed un rallentamento nella frequenza di interazione con il cellulare. Quindi utilizzando tutte queste informazioni son riusciti a creare un modello che da ottimi risultati.

Infine c'è la tesi di Marco Spallone [11], su cui si basa il mio progetto. Il lavoro affronta il medesimo problema di [10] utilizzando la tecnica *object detection*, in italiano *rilevamento degli oggetti*. Il rilevamento degli oggetti è una tecnologia informatica legata al computer vision e all'elaborazione delle immagini che si occupa di rilevare istanze di oggetti semantici di una certa classe (come esseri umani, edifici o automobili) in immagini e video digitali. L'autore utilizza la tecnica sopracitata per riconoscere elementi comuni a tutte le macchine, con lo scopo di discernere se la fotografia rappresenta il lato sinistro o destro dell'abitacolo. La tesi di Spallone utilizza questa tecnica per individuare due oggetti, cioè la cintura ed il finestrino.

Il progetto che viene illustrato si basa sulla tesi [11] e ha l'obiettivo di colmare alcune sue criticità:

- il modello utilizza servizi esterni per interrogare il modello, rendendo il progetto dipendente da essi;
- i test sono stati condotti in due fasi separate: la prima in cui si sono analizzati tutti i frame di un video, la seconda dove si sono elaborati i risultati. Tuttavia in un caso reale le due fasi devono essere eseguite in modo sequenziale, per ogni frame;

- l'impossibilità di avviare i test utilizzando solamente lo smartphone;
- la mancanza applicazione in una situazione reale
- la necessità d'utilizzo di internet per avviare i test.

Il progetto su cui è basata questa tesi, oltre a risolvere tali criticità, è stato sviluppato con l'intento di creare una base per progetti futuri e per semplificare la fase di *testing* del modello. Infatti l'applicazione permette al suo utilizzatore di testare il modello con i parametri scelti dall'utente. L'approccio al problema presenta delle differenze rispetto a quello di [11].

- l'analisi condotte nelle finestre scorrevoli presenta un'euristica diversa;
- gli output del modello vengono analizzati immediatamente aggiornando i dati ad ogni frame acquisito;
- le immagini sono acquisite ad una distanza di tempo Δt che descrive la quantità di tempo necessaria per fare l'analisi di un frame;
- le foto sono catturate in tempo reale.

Grazie a questo diverso approccio, si rilevano i seguenti vantaggi:

- con Δt le immagini presenti nelle finestre scorrevoli sono più eterogenee rendendo la classificazione più vicina alla realtà;
- la metrica riguardante il tempo rispecchia i tempi necessari all'applicazione, oltre al modello, a raggiungere una soluzione;
- i test corrispondono ad una situazione reale;
- i test possono essere condotti da chiunque, anche da un utente non esperto;
- la creazione del dataset è stata realizzata da più persone, rendendolo più eterogeneo;
- il modello è stato allenato usando le risorse locali;

- il modello si trova nel cellulare pertanto non è necessario interrogare servizi esterni e l'applicazione è totalmente indipendente;
- il modello si trova nel cellulare pertanto non è necessario l'utilizzo di internet o di servizi esterni;
- la possibilità di calcolare con più metodi diversi, e di poter comparare i risultati tra loro;
- la creazione di un servizio utilizzabile in una situazione reale.

Capitolo 2

Architettura

In questo capitolo si tratterà delle più significative scelte architetturali compiute durante l'implementazione dell'applicazione. Di seguito la definizione delle parole chiave maggiormente utilizzate:

Model : il modello machine learning utilizzato per definire le previsioni.

Threshold : il limite da superare affinché possa essere considerato valido il dato.

Window : una finestra di valori (frame in questo caso) all'interno della quale si elaborano i calcoli necessari.

Frame : istantanea del video che viene dato in input al modello.

Client : la classe dal lato UI che interagisce direttamente con il servizio che gestisce il modello.

Valutazione : un test completo, comprensivo di uno o più frame. Una volta completata la valutazione, si ottengono il gruppo finale e le statistiche sulla stessa.

Classificazione : è il nome attribuito all'oggetto identificato dal modello.

Gruppo : è un'astrazione superiore rispetto alla classe, per suddividere le classi in "Passenger" o "Driver".

Confidence : è la percentuale di correttezza del dato.

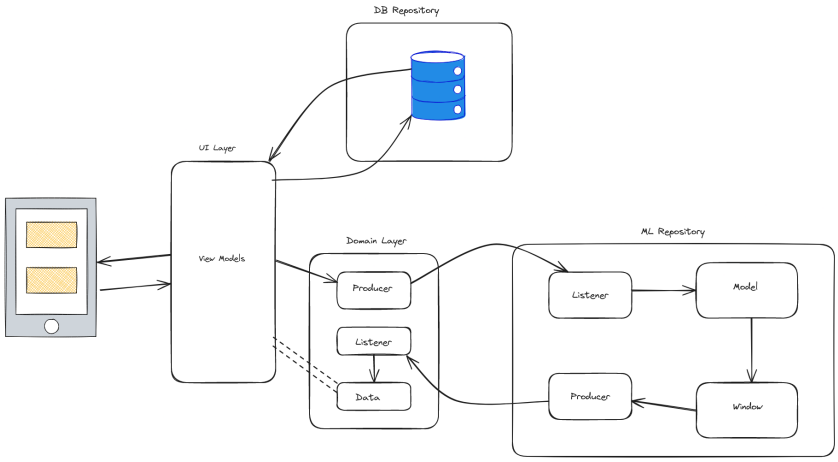


Figura 2.1: Semplificazione dell'architettura del progetto

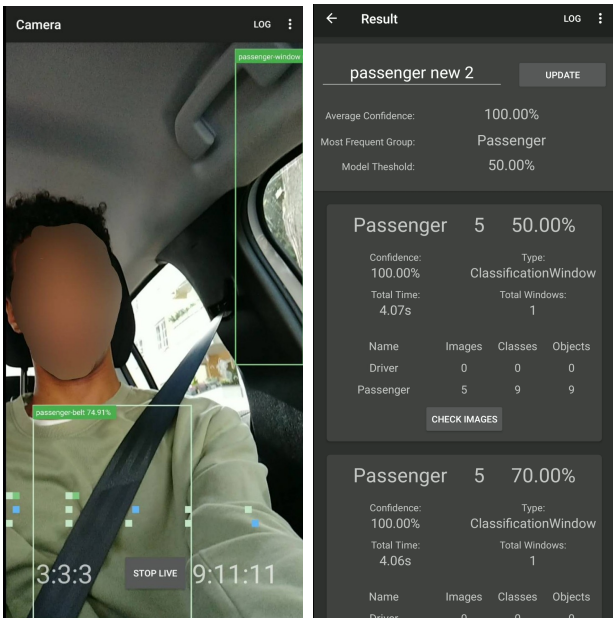
Bounding Box : il rettangolo che racchiude al suo interno l'oggetto identificato dal modello. Altri valori possono essere associati ad esso quali la Confidence e la classificazione.

2.1 Funzionamento

L'applicazione ha due funzionalità principali, la prima per testare il modello e verificarne l'efficacia in base a dei parametri stabiliti dall'utente, la seconda per navigare i test passati. Nella figura 2.1 vengono rappresentati i flow delle due funzionalità: il primo che passa attraverso *ML Repository*, il secondo che interroga il database.

Testing

Per testare il modello l'utente dovrà premere sul tasto start: il modello inizierà a raccogliere ed analizzare i frame distribuiti dalla libreria Camera2. Un frame viene raccolto solo alla fine della valutazione del fotogramma precedente. Questo fa sì che trascorra una certa quantità di tempo tra un frame e il successivo, rilevando quindi input più eterogenei ed esprimendo di conseguenza una soluzione più valida. Il modello quindi rilascerà un output per ogni foto in input, il quale verrà analizzato dalla Window ed infine visualizzato dall'utente. L'utente riuscirà a vedere ogni oggetto che è stato rilevato



(a) Rappresentazione dell'applicazione nella fase di valutazione

(b) Rappresentazione dell'applicazione nella visualizzazione dei risultati

Figura 2.2: Screenshot dell'applicazione nelle due attività principali

ed un'aggregazione delle metriche riguardanti lo stesso in tempo reale, permettendo di percepire il funzionamento del modello. Inoltre, una volta terminata positivamente la valutazione, potrà visualizzare i dati per ogni singola composizione dei parametri espressi inizialmente.

Visualizzazione test passati

Per quanto riguarda la seconda funzionalità, si pescheranno i dati precedentemente salvati nel database, in modo che possano essere visualizzati come al termine della valutazione in tempo reale.

2.2 Single Activity Pattern e gestione dell'UI

La parte grafica dell'applicazione è suddivisa in base alle funzionalità cioè con la stessa modalità descritta nel paragrafo 2.1. Per ottimizzare la quantità di codice reiterata si è deciso di utilizzare il Single Activity Pattern, il quale prevede l'utilizzo di una singola activity nell'intera l'applicazione. Questo tipo di pattern ci permette di limitare la ripetitività del codice e di poter condividere dei dati con tutto il resto dell'applicazione.

La condivisione dei dati avviene grazie alla ViewModel legata all'activity e all'entità Client. Nel caso della ViewModel è possibile poiché essa è legata al lifecycle dell'activity, e dato il pattern descritto precedentemente si può comprendere che i dati e le variabili persisteranno fino alla terminazione dell'applicazione. Nel secondo caso invece è possibile poiché il Client è un singleton, cioè esiste una sola istanza di quella classe in tutta l'app. Pertanto utilizzando questi oggetti è possibile seguire e mantenere il pattern dei cinque principi SOLID.

Camera Fragment

Questo fragment gestisce e visualizza la valutazione in real time, infatti grazie ad esso è possibile visualizzare i *bounding box* nello schermo ed analizzare i risultati una volta completato il processo. Da questo elemento è possibile navigare al SettingsFragment che permette di modificare i parametri legati alla valutazione.

Log Fragment

Questo fragment permette di visualizzare i risultati passati e rappresenta l'unica sezione che interroga il database per acquisire i dati. Il modo in cui vengono rappresentati è lo stesso descritto nel camera fragment per poter riutilizzare il codice. Per ogni test si vedono i risultati di ogni singola finestra, per ogni finestra quali sono i frame considerati e quali le classi riconosciute. Inoltre per ciascun frame si può visualizzare la foto con gli effettivi oggetti riconosciuti.

2.3 Il Flow dei test

Per eseguire i test si è dovuto sviluppare un sistema asincrono basato sul pattern client-server. Il client e il server innanzitutto creano una connessione, un sistema binario in cui l'uno può essere registrato come ascoltatore dello stream di dati dell'altro, dopodiché inizia il vero e proprio scambio di dati finché la connessione non viene interrotta. La comunicazione avviene attraverso la descrizione degli stati Ready, Start, Input/Output, End, i quali, per le due entità, sono pressoché i medesimi. In seguito la descrizione degli stati citati.

Ready : comunica che l'entità è pronta per ricevere nuovi stati;

Start : richiede o conferma l'inizio della valutazione;

Input/Output consegna un input o ritorna un output in base all'entità;

End richiede o conferma lo stop della valutazione. Nel caso del server viene aggiunto anche il risultato finale.

Per non dover gestire la possibilità di avere più input contemporaneamente si permette al server di ascoltare un solo stream in entrata e, al contempo, si dà l'opportunità a più entità di ascoltare il suo flusso di dati, in modo affine ad un broadcast. Ciò ci permette di suddividere gli ascoltatori dello stream in base al loro scopo, nel nostro caso abbiamo una classe che gestisce e mantiene tutti i dati del flusso e un'altra che sincronizza gli stati UI con quelli dell'output.

Per poter definire tale comunicazione tra il client e il server si sono definite due classi:

Listener : per poter ascoltare un flusso di dati;

Producer : per trasmettere dati.

Pertanto, analizzando questo pattern nella nostra applicazione, è possibile identificare ImageDetectionRepository come server e il Client come il client che produce input e ascolta e mantiene gli outuput. In questo caso si è deciso di implementare il produttore di input solo nel client per evitare di dare la possibilità ad altre classi di inviare input al ImageDetectionRepository.

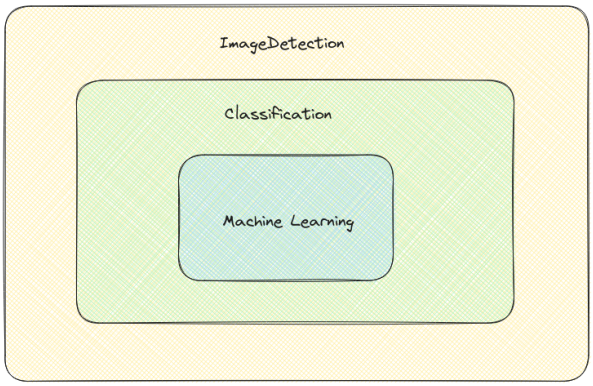


Figura 2.3: Dimostrazione dei vari livelli di astrazione

2.4 Componenti

Sono stati definiti tre possibili livelli di astrazione per semplificare il riuso del codice e per suddividere le funzionalità in base al loro scopo. Ogni classe appartenente ad un determinato livello può utilizzare solo variabili e dati del proprio livello o minore. L'utilizzo di questo tipo di astrazione aumenta drasticamente la flessibilità del progetto, infatti se per esempio si vuole creare un nuova classe che rappresenta un modello che utilizza la tecnica *object detection*, è sufficiente estendere l'ultimo livello. Se si vuole creare un modello che usa un tecnica diversa ma presenta delle classificazioni, si può estendere il livello *Classification*. Oltre ad utilizzare la logica di classi preesistenti, si può fare affidamento ad una struttura ben consolidata che rispecchia le proprie esigenze.
I livelli descritti sono:

Machine Learning : questo è il livello primario in cui ci si aspettano i dati più grezzi e di base. In generale viene considerata solo la confidence;

Classification : quando il dato presenta anche una classificazione, con le statistiche e metriche collegate ad essa;

Image Detection : l'ultimo livello è rappresentato dalla descrizione effettiva del type dell'input e output, con la possibilità di effettuare delle scelte più dettagliate in base alle classi su cui vengono applicate.

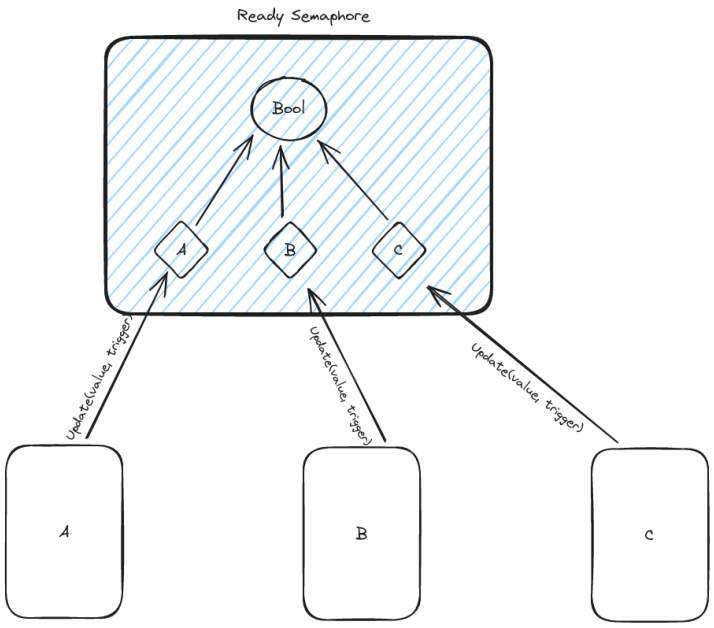


Figura 2.4: Esemplificazione del funzionalmento della classe ReadySemaphore all'interno della repository

Per ognuna delle componenti sottostanti sono stati applicati i seguenti tre livelli di astrazione.

2.4.1 ImageDetectionRepository

Questa classe è molto particolare perché fa da gateway verso l'esterno e allo stesso momento gestisce le varie componenti interne. Al suo interno si trovano il Model ed il WindowManager. Per organizzare e gestire le varie componenti interne sono state implementate delle classi specifiche: la più evidente è il ReadySemaphore che serve a reagire ai cambiamenti degli stati del Client, del Model e delle Preference. L'altra classe degna di nota è il LiveEvaluationProducer la quale si prende in carico la trasmissione dei vari stati del sistema. Grazie all'utilizzo di queste due classi si è potuto minimizzare le righe di codice replicato nei vari livelli.

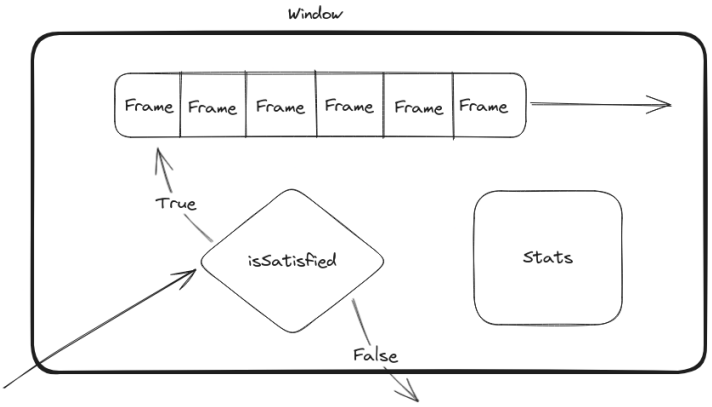


Figura 2.5: Dimostrazione del funzionamento di una finestra

2.4.2 WindowManager

La necessità di questa classe deriva dall'idea di utilizzare molte Window differenti contemporaneamente. Queste finestre seguono le preferenze dell'utente, in modo da poter dimostrare le loro differenze o similitudini utilizzando lo stesso sample di input. All'interno della classe si ha una factory per riprodurre tutte le Window con le combinazioni delle preferenze. I parametri di un Window possono essere:

- S: Insieme delle lunghezza della finestra.
- Th: Insieme delle soglie di confidence necessarie per completare il Window.
- Ty: Insieme delle tipologia di finestra, questo parametro decide il tipo di euristica da utilizzare per accettare i frame.
- O: Insieme delle quantità di immagini da accettare prima di poter considerare un Window completabile.

Considerando W l'insieme di combinazioni del Window si ha che $|W| = |S| * |Th| * |Ty| * |O|$. Se si considera che una finestra è considerata la combinazione di questi parametri, si può evincere che la quantità di Window può incrementare molto rapidamente. Pertanto i compiti principali del WindowManager sono la creazione e coordinazione delle Window, oltre alla gestione dei dati aggregati.

2.4.3 Window

Una Window può essere rappresentata come una finestra di valori con una grandezza statica, nel gergo informatico può essere vista come una coda FIFO (First Input First Output) con dimensione fissa. Ogni window rappresenta una combinazione delle preferenze dell'utente e conterrà al suo interno tutte le statistiche e metriche di tale combinazione. Ogni window contiene una condizione di completamento: una volta che questa condizione diventa positiva, vengono raccolti e analizzati tutti i dati. A questo punto la window non accetterà alcun input finché non verrà ripristinata la condizione iniziale.

2.4.4 Client

Questa classe fa da tramite tra il UI e la repository. Si tratta di un Singleton, vale a dire, un design pattern creazionale con lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. L'obiettivo di questo oggetto è quello di mantenere e manipolare i dati raccolti durante la valutazione. Pertanto durante la valutazione è possibile interrogare questa classe per acquisire i dati aggiornati in tempo reale. Poi, una volta completata la valutazione correttamente, vengono copiati e memorizzati i dati per potervi accedere fino al completamento della valutazione seguente.

2.4.5 Model

Il modello ha molti più livelli di astrazione rispetto alle altre classi, poiché il terzo livello è stato suddiviso a sua volta in ulteriori livelli. Il terzo livello crea un'astrazione in base al tipo di sorgente di dati o libreria utilizzata. Nel nostro caso viene sviluppato un modello basato sulla libreria PyTorch ma si possono utilizzare altre fonti quali librerie diverse o modelli interrogabili tramite internet. In generale il modello ha tre funzioni a catena:

- *preprocess*: funzione che si occupa di manipolare l'input per renderlo congruo alle specifiche del modello;

- *evaluation*: effettiva valutazione dell'input la quale ritornerà l'output grezzo;
- *postprocess*: funzione finale per processare l'output in modo da renderlo leggibile da parte del suo utilizzatore.

Oltre a queste tre funzioni il modello contiene anche una classe interna Producer per poter comunicare al suo utilizzatore lo stato interno del modello. Questi sono gli elementi presenti nel livello base, poi per ogni astrazione superiore ci saranno delle funzioni in più e più specifiche. Il fatto di aver creato queste tre funzioni a catena che utilizzano il risultato della funzione antecedente si possono facilmente legare mimando un flow di dati. Il modello è anch'esso un singleton, risiede unicamente nella repository ImageDetectionRepository e costituisce il la sorgente di dati di tale repository.

2.5 Gestione del database

Per mantenere tutti i dati dei vari test salvati, si è deciso di utilizzare il database consigliato da Google chiamato Room. Questo database è un wrapper per il vero database presente che è SQLite. I componenti principali di Room sono tre:

- la classe del database, che contiene il database stesso e serve come punto di accesso principale per la connessione ai dati persistenti dell'applicazione;
- le entità di dati che rappresentano le tabelle del database dell'applicazione;
- gli oggetti di accesso ai dati (DAO) che forniscono metodi che l'applicazione può utilizzare per interrogare, aggiornare, inserire e cancellare i dati nel database.

La classe del database fornisce all'applicazione istanze dei DAO associati al database. A sua volta, l'applicazione può utilizzare i DAO per recuperare i dati dal database come istanze degli oggetti entità associati. L'applicazione può anche utilizzare le entità definite per aggiornare le righe delle tabelle corrispondenti o per creare nuove righe da inserire.

La tabella principale è *Evaluation* la quale rappresenta l'oggetto base. In questa tabella 2.6 vengono salvate le caratteristiche e statistiche principali della valutazione, ad esempio la confidence media o il threshold utilizzato dal modello per una determinata

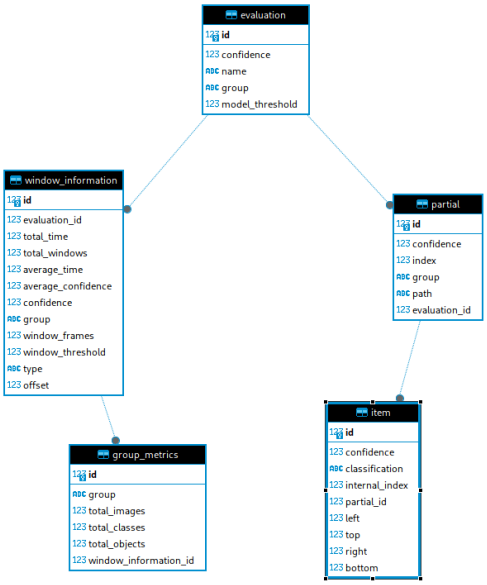


Figura 2.6: Rappresentazione dello schema finale del database

entità. Evaluation ha due tabelle associate a se, con cui ha un relazione uno a molti. La prima di cui parleremo è quella riguardante le informazioni del Window, ci sono tutte le metriche e le statistiche per ogni singola finestra appartenente alla Evaluation; per ognuna di esse ci sono anche le statistiche riguardanti ogni singolo gruppo e classe analizzati dalla Window. Queste informazioni verranno utilizzate per formulare i risultati finali. Se invece si indaga nella tabella dei *Partials*, si troveranno in generale i dati riguardanti le soluzioni parziali come il path per l'immagine salvata, l'indice della foto rispetto alla Evaluation e la confidence media. Legato ai Partials c'è Item che rappresenta ogni classe che è stata trovata in ciascun Partial. In Item troviamo i valori del quadrato rispetto alle dimensioni dell'immagine, la classe e la confidence.

Capitolo 3

Implementazione

In questo capitolo si porrà il focus su alcune scelte implementative del progetto, su come sono state sviluppate e sulla loro fuzionalità nell'applicazione.

Il linguaggio utilizzato per l'implementazione dell'applicazione è Kotlin. Kotlin è un linguaggio relativamente nuovo, noto come successore di Java e che, come quest'ultimo, utilizza la JVM per compilare i propri file. Kotlin è riuscito a definire dei concetti nuovi che in Java non sono presenti, idee che sono in gran parte implementate da Librerie esterne. Un esempio sono i Flow: in Kotlin sono una componente nativa mentre in Java vengono sviluppati tramite la libreria RxJava.

3.1 Producer e Listener

Queste due classi sono state largamente utilizzate nel progetto, la classe base è relativamente semplice. Ogni oggetto Producer contiene al suo interno un oggetto SharedFlow che è il tipo di stream più generale, di cui si ha la possibilità di specificarne i parametri, compreso la CoroutineScope, nel costruttore della classe stessa. Il Producer dà inoltre la possibilità di osservare i dati interni, grazie ad una variabile ObservableData, che permette agli oggetti che li osservano di reagire ad ogni cambiamento di stato. Si è deciso di creare due wrapper delle funzioni del SharedFlow, cioè *tryEmit* ed *emit*, che sono essenziali per gestire al meglio il problema della colorabilità delle funzioni.

Diversamente, al Listener viene passato il SharedFlow nel costruttore e l'unico suo compito è collezionare tutti i dati che gli vengono inviati. Anche questa classe mette a disposizione una variabile ObservableData per poter osservare i cambiamenti di stato.

```
1    override fun listen (scope: CoroutineScope, inputFlow: SharedFlow<S
     >?, mode: IGenericMode) {
2        destroy()
3
4        job = scope.launch(Dispatchers.Default) {
5            when (mode) {
6                GenericMode.First -> if (!inputFlow?.replayCache.
     isNullOrEmpty()) mCurrentState.update(inputFlow?.replayCache?.first
     ())
7                GenericMode.Last -> if (!inputFlow?.replayCache.
     isNullOrEmpty()) mCurrentState.update(inputFlow?.replayCache?.last()
     )
8                GenericMode.None -> {}
9            }
10           inputFlow?.collect { state -> collectStates(state)}
11       }
12    }
```

Essendo entrambe le classi astratte, per utilizzarle è necessario estenderle. Ogni derivata di queste due classi dovrà definire delle funzioni per poter gestire le specifiche situazioni.

Per ognuna delle due classi è stata inoltre definita un'altra derivata per poter acquisire o emettere i dati in modo atomico, infatti se una di queste azioni viene compiuta troppo rapidamente, c'è il rischio che l'azione non vada a buon fine e venga scartata. Per ovviare al problema si è utilizzato un Mutex, rendendo la risorsa accessibile atomicamente.

3.2 Astrazione

L'utilizzo di più livelli di astrazione nel back-end del progetto è stata decisa per aumentare la flessibilità del progetto e la sua elasticità. L'estensione di queste classi è molto semplice, infatti sono state definite affinché si possano utilizzare con la minima quantità di codice. L'obiettivo di questa scelta è di consentire a possibili studenti futuri di

poter estendere l'applicazione avendo già una base solida da cui partire. Infatti, nel caso in cui uno sviluppatore voglia aggiungere altre funzionalità legate al MachineLearning, come l'utilizzo di un nuovo modello o lo studio di altri processi non legati al rilevamento di oggetti in un'immagine, egli avrà la possibilità di estendere tutte le classi del primo o del secondo livello se vi è l'intenzione di classificare l'output.

Questo tipo di approccio molto utilizzato nella filosofia OOP è stato possibile poiché Kotlin permette di usare i Generic. Per le quattro entità principali, cioè Client, Repository, Model e Window, sono stati utilizzati quattro Generic principali: il primo identifica l'input, il secondo l'output parziale, il terzo serve per descrivere il risultato finale e l'ultimo per definire come identificare una classificazione. Infatti l'ultimo Generic viene aggiunto solamente nel secondo livello, quello riguardante la classificazione. Quest'ultima Generic nel progetto sarà sempre una stringa, in casi più complessi può diventare una classe non nativa.

3.2.1 Astrazione dei dati

Una domanda che può sorgere è: come fanno queste entità a comunicare tra loro? Infatti, l'utilizzo di questi livelli di astrazione ci ha portato anche a definire dei dati con la stessa complessità. Ogni possibile classe che descrive un dato è stata definita per ogni tipo di livello e questo dato viene rappresentato come un oggetto immutabile. Dato che in Kotlin, nel momento in cui si passa un oggetto non nativo lo si fa sempre per valore, con l'utilizzo di un oggetto immutabile si possono condividere le informazioni sapendo che il suo utilizzatore non potrà modificarne il valore. Infatti, per modificare un oggetto immutabile è necessario creare prima una sua copia per poi applicare le modifiche ad essa.

3.3 Cambiamento delle preferenze

Il cambiamento dei topic è stato una modifica relativamente difficile poiché si è dovuto tener conto di molte variabili. Innanzitutto è bene descrivere che cosa sono le preferenze. I setting, nel termine inglese, sono suddivisi in due: quelli che modificano i parametri del modello e quelli impiegati per scegliere i valori delle finestre. Una volta selezionati,

questi vengono salvati in variabili ed una volta usciti dal SettingFragment vengono salvati nel dataStore. Seguendo il pattern SSOT, la repository del riconoscimento di immagini prende in input lo stream di dati messo a disposizione dal DataStore e, in questo modo, può aggiornare i propri dati in risposta ai cambiamenti dello stato del dataStore stesso. Le modifiche apportate al modello sono le più semplici e richiedono il cambiamento di una sola variabile mentre quelle per la Window sono più complesse e richiedono delle ottimizzazioni.

Le Window possiedono 4 parametri che vengono passati nel costruttore, questi valori sono:

- **O**ffset: la minima quantità di Window prima di poter considerare la finestra terminabile;
- **S**ize: la grandezza della finestra;
- **Th**reshold: il limite di precisione che la Window deve avere affinché possa essere soddisfatto;
- **Ty**pe: la tipologia di window utilizzata.

Per calcolare la quantità di finestre stanziate, è necessario contare tutte le possibili combinazioni di valori dei parametri cioè $S * Th * Ty * O$. Come si può facilmente intuire, la quantità di Window incrementa molto rapidamente. Come esempio pratico, per condurre i test di questo progetto sono state utilizzate in tutto 540 Window. Quindi risulta che il cambiamento dei vari parametri porterà alla creazione e/o distruzione di Window.

```
1     fun update (newSettings: IWindowSettings) {
2         // considering the old set as Curr and the new one as New
3         // get the list of settings as a set and get all the windows
    that are not part of the current ones
4         val listOfNewSettings = newSettings.asListOfSettings().toSet()
    as Set<S>
5
6         // create |New/Curr| windows
7         val newWindows = factory.createMapOfWindow(listOfNewSettings.
    minus(currentWindows.keys))
```

```
8
9         // remove the windows not part of the new settings and add the
      one that are not there
10        currentWindows = currentWindows
11            // Curr = Curr intersect New
12            .minus(currentWindows.keys.minus(listOfNewSettings))
13            // Curr = Curr union New
14            .plus(newWindows)
15
16        // update the active set of windows
17        activeWindows = currentWindows.values.toSet()
18    }
```

Andando a considerare il metodo di implementazione più naive, cioè con la distruzione di tutte le finestre attuali e la creazione di nuove finestre con parametri diversi, si può rilevare un dispendio di risorse e tempo molto alti. Ciò poiché, nonostante si modifichi il valore di una sola preferenza, si ha la distruzione e nuova creazione di tutte le Window, anche quelle non necessarie. Per ottimizzare l'algoritmo si è deciso di usare un approccio diverso, cioè di considerare le combinazioni di setting al pari di insiemi. Considerando Pa come preferenze attuali e Pi come nuovo input di preferenze, il nuovo metodo presenta due azioni principali: la prima è l'eliminazione delle Window in sovrappiù, che è rappresentata dall'insieme risultante da $Pa - Pi$ mentre la seconda è la creazione delle nuove finestre $Pi - Pa$. Quindi la differenza tra il primo ed il secondo algoritmo è che non si svolgerà alcun tipo di operazione nell'intersezione dei due insiemi $Pi \cap Pa$.

3.4 Gestione delle classificazioni

Le classificazioni vengono gestite da una classe chiamata Classifier. Questa classe è considerata il *data source* delle classificazioni. Non si è ritenuto necessario definire una repository che gestisse questa fonte di dati essendo utilizzata solamente dal modello, ma in futuro se si utilizzerà una struttura diversa, per esempio legata al database, sicuramente sarà considerato il metodo migliore. Nel progetto le classificazioni vengono acquisite da un file JSON, il quale descrive per ogni gruppo le classi ad esse collegate. Una volta serializzato, il file viene riconvertito in un Map, che è una struttura di dati

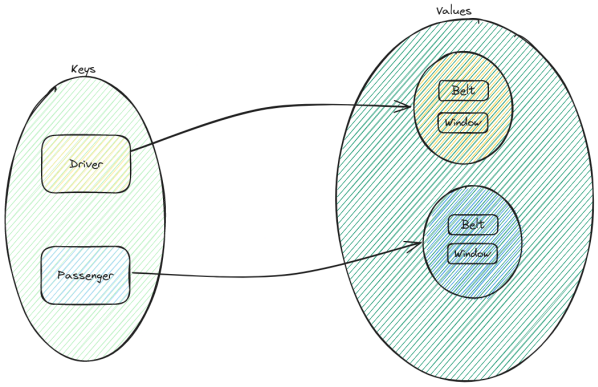


Figura 3.1: Struttura dati del *Classifier*

che presenta un insieme di chiavi e per ogni chiave è associato un valore. Quindi questa classe salva i dati utilizzando come chiave il gruppo e come valore un insieme di classificazioni. Il Classifier viene utilizzato unicamente dai modelli dal secondo livello in su. Nel nostro caso viene utilizzato per assicurarci che le classi trovate dal modello siano valide e, dato che la creazione di questi gruppi è un'astrazione che non appartiene al modello, ci permette di avere un collegamento tra questi due dati.

3.5 Riconoscimento degli oggetti e loro visualizzazione

Nell'interrogazione di un modello si ha che il modello ritorna un output che non è altro che un array di Float. Per comprendere quali sono i dati all'interno di questo array si può rappresentare il vettore come se fosse una tabella avente un numero a di colonne. Questo numero è calcolabile come $a = c + o$, dove c è il numero di classi e o è una tupla *(w, y, w, h, confidence)*. I primi due valori corrispondono alle coordinate del centro del rettangolo rispetto all'immagine, w e h sono invece rispettivamente la lunghezza e l'altezza del rettangolo mentre l'ultimo valore, come si può facilmente intendere, è la confidence relativa al *bounding box*. I dati presenti nelle colonne che interessano le classi rappresentano invece la percentuale di possibilità che la bounding box sia quella particolare classe. Successivamente, mettendo assieme tutti questi dati, si può costruire

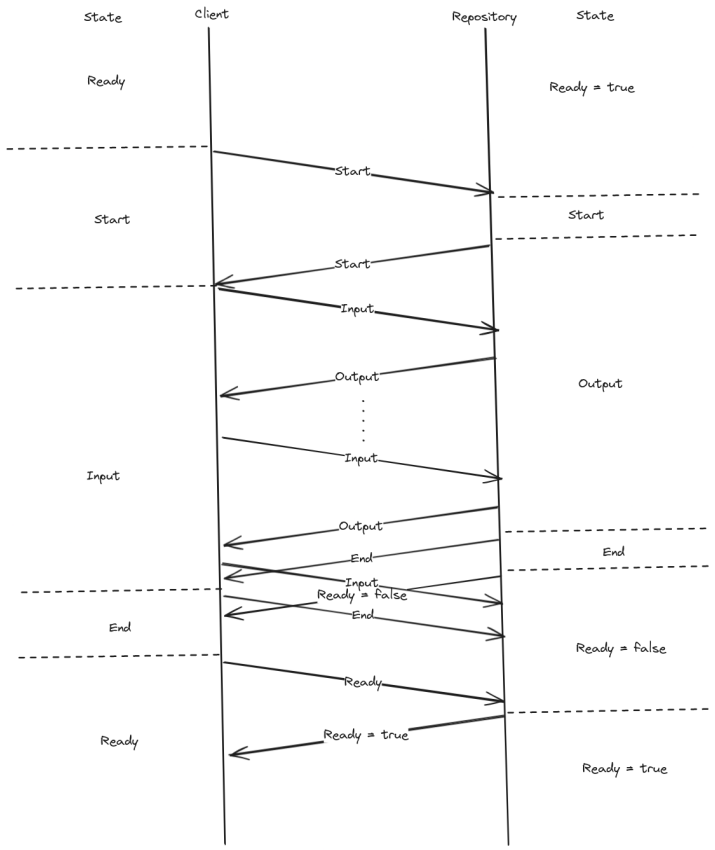


Figura 3.2: Esempio del flusso di dati durante una valutazione completa tra Client e Repository

un rettangolo che rappresenta l'oggetto, e associarvici una classe e la sua confidence.

3.6 Descrizione della communicazione tra Repository e Client

Pertanto, come avviene il flusso di dati? Innanzitutto partendo dal CameraFragment, il quale dà la possibilità di compiere l'azione "start" solo nel caso in cui sia la repository che il Client siano nello stato *Ready*. A questo punto, premendo Start, la classe Client richiede l'inizio della valutazione e la Repository replica con lo stesso stato per confermare; la classe che gestisce la fotocamera acquisisce un fotogramma e lo consegna al Client;

Model	*size*	*mAP*	*SpeedCPU*	*SpeedA*100	*params*	*FLOPs*
*yolov*5*nu*	640	34.3	73.6	1.06	2.6	7.7

Tabella 3.1: Prestazioni del modello YOLOv5n

quest'ultimo lo invia alla repository attraverso lo stream di dati e attende la sua risposta; infine il server riceve il frame e lo fornisce al modello. Successivamente, se il gestore di Windows riconosce che tutte le finestre hanno raggiunto una conclusione, la Repository invierà al Client anche lo stato End con il risultato finale di tutta la valutazione. Nel caso contrario, la Repository aspetterà il prossimo input e così via. L'unica possibilità per la valutazione di non completare l'intero ciclo è che ci sia un errore interno alla Repository la quale invia uno stato End con un messaggio generale di errore; oppure per richiesta da parte del Client perché l'utente potrebbe aver deciso di terminare la valutazione o di uscire dall'applicazione. In quest'ultimo caso verrà mandato al medesimo lo stato End ma con un errore vuoto. Al termine della valutazione la Repository aspetterà che il Client invii lo stato "Ready" e a quel punto, se tutti i sistemi interni della Repository sono pronti, invierà anch'esso lo stato "Ready", ripristinando la situazione iniziale.

3.7 Modello

Il modello utilizzato per il progetto è ***YOLOv5***, ottimizzato per la rilevazione (detection) di oggetti. E' stato scelto poiché ha un'ottima documentazione ed è perfettamente integrato nella libreria ***PyTorch*** da noi utilizzata. E' inoltre lo stesso modello utilizzato nella tesi [10], dandoci la possibilità di comparare i risultati ritenuti interessanti.

Per questo progetto si è deciso di utilizzare un modello pretrainato cioè con dei pesi già presenti, chiamati yolov5n. Con riferimento alla performance è considerato il più veloce ma con un conseguente abbassamento nella qualità di riconoscimento degli oggetti, come evidenziato nella Tabella 3.1.

3.7.1 Dataset

Raccolta delle immagini

Il dataset del progetto è stato costruito con 326 immagini, catturate dal sottoscritto e da conoscenti per poter avere un insieme di foto più eterogeneo possibile. Le foto sono state scattate durante il giorno e la notte e, in ogni immagine, sono visibili la finestra laterale e la cintura. Le foto sono equamente distribuite per gruppo, dunque si avranno 163 foto per il guidatore e 163 foto per il passeggero.

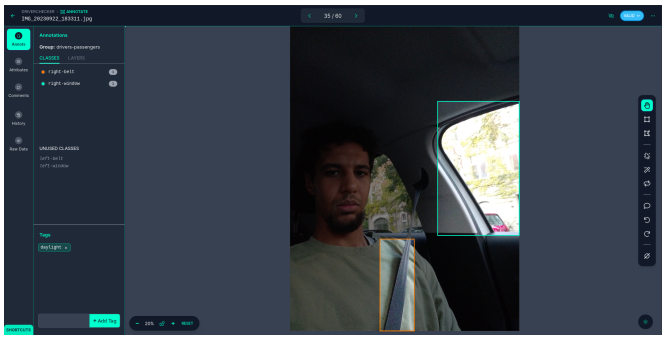
Labeling

A questo punto inizia la fase di *labeling*. Questa fase è particolarmente importante perché per ogni immagine si va a definire manualmente la classificazione degli oggetti creando un rettangolo attorno ad essi. Questa fase è delicata perché se mal progettate possono portare il modello a diventare overfitted o underfitted [14].

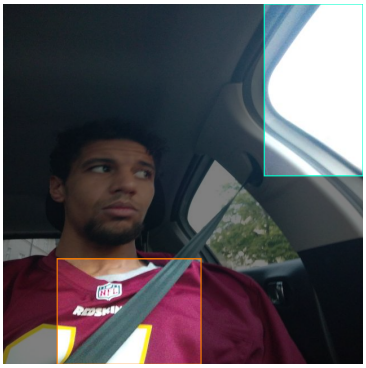
Completamento del dataset

simb Successivamente queste fotografie passano attraverso altri due fasi. La prima fase si chiama *preprocessing* e consente di ritagliare, ridimensionare e ruotare l'immagine prima del training per renderla conforme alle necessità del modello. Per il modello *YOLOv5*, in particolare, viene richiesto che ogni immagine sia nel formato *640x640*. Il secondo livello è chiamato *augmentation* ed esegue trasformazioni sulle immagini esistenti per crearne nuove varianti e aumentare il numero di immagini nel set di dati. In questo modo i modelli diventano più precisi in una gamma più ampia di casi d'uso. Per applicare questa tecnica viene utilizzato il framework Roboflow [15], il quale permette di cambiare le proprietà nelle modalità che si vuole. Nel nostro caso si è deciso di applicare:

- Rotazione: fra -15° e +15°
- Luminosità: fra -25% e +25%
- Esposizione: fra -25% e +25%



(a) Fase di *labeling* con il framework *Roboflow*



(b) Esempio di augmentazione di una immagine

Figura 3.3: Screenshots delle varie fasi della definizione del dataset

Grazie a questo passaggio le immagini totali diventano 808. La quantità minima di immagini consigliate dalla compagnia che possiede YOLOv5 è di 1.500 immagini per classe e circa 10.000 istanze per ogni classe, cioè, all'interno del dataset, si vogliono 1.500 immagini avente almeno un oggetto con tale classe e la presenza di circa 10.000 oggetti con la medesima classe. Nel nostro caso ce ne sono circa 400 immagini per ogni classe e circa 800 istanze totali per classe, ma, essendo un progetto con l'obiettivo di dimostrare la sua utilità, può considerarsi una quantità adeguata. Il set di dati, infine, viene suddiviso in tre categorie, le quali verranno utilizzate nella fase seguente. La suddivisione delle immagini in *Train*, *Evalutation* e *Test*, permette di consolidare i risultati ottenuti durante questa fase perché oltre alle immagini per allenare il modello, ce ne sono altre per convalidarlo e per testarlo. Questa modalità verrà utilizzata per ogni epoch, cioè per ogni ciclo.

La creazione del dataset è molto importante poiché migliore è il dataset e più accurato sarà il riconoscimento da parte del modello nelle possibili variazioni. Un dataset non progettato adeguatamente può portare ad un modello sbilanciato, che presenta cioè difficoltà nel riconoscere alcuni oggetti rispetto ad altri. Per questo la creazione del dataset viene considerata un'operazione particolarmente delicata.

3.7.2 Training

Dopo aver costituito il dataset, inizia la fase di ***training***. In questa fase si utilizza il dataset per modificare i pesi presenti nel modello. In generale maggiore è il numero di ***epoch*** (cicli) e migliore sarà il risultato finale. La fase di training è molto delicata: ad esempio nel caso in cui non si allena abbastanza il modello si può rischiare di avere come risultato un prodotto sbilanciato cioè allenato solo per riconoscere le immagini e/o situazioni che sono parte del dataset. Per eseguire questa fase si è utilizzato il framework open source di Ultralytics che sviluppa alcuni comandi dal terminale per gestire il modello. Questa applicazione permette di attuare tutte le operazioni utili a manipolare le fasi successive alla creazione del dataset completo. Nel nostro caso viene utilizzata per il training e per la conversione del modello in un formato leggibile per il mobile, come verrà spiegato più avanti. L'allenamento del modello è stato eseguito con il database descritto sopra, per 50 epochs totali.

3.7.3 Risultati e Statistiche

Il risultato ottenuto è migliorabile, ma certamente è abbastanza buono per il nostro caso. L'applicazione utilizzata permette di visualizzare i dati sia in tabella che tramite grafici.

Per leggere correttamente i grafici presenti nella figura 3.4 è necessario conoscere il significato corretto dei termini utilizzati:

mAP : confronta il riquadro di delimitazione della verità assoluta con il riquadro rilevato, e restituisce un punteggio. Più alto è il punteggio, più il modello è accurato nei suoi rilevamenti;

box_loss : è la correttezza del rettangolo in cui è compreso l'oggetto rispetto alla verità;

cls_loss : è la correttezza delle classificazioni rispetto alla verità;

obj_loss : è la correttezza degli oggetti rispetto alla verità;

precision : è la frazione di istanze rilevanti tra le istanze recuperate;

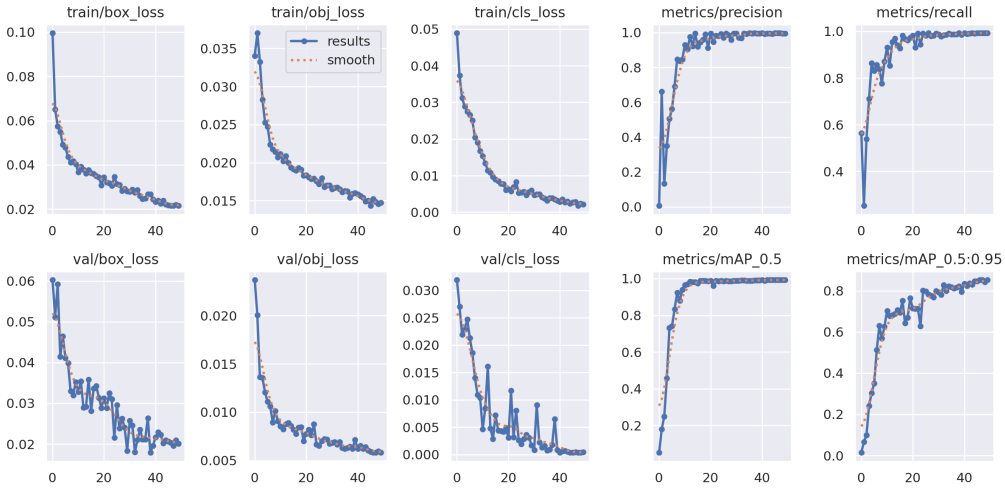


Figura 3.4: Risultati del training del modello

recall : è la frazione di istanze rilevanti che sono state recuperate.

Dai grafici nella figura 3.4 si può facilmente comprendere che il modello è bilanciato correttamente.

3.7.4 Export

L'ultimo passaggio prima dell'utilizzo del modello, nell'applicazione, è la conversione ad un formato leggibile per il tipo di device utilizzato. Il framework utilizzato, PyTorch, richiede il format ".torchscript", inoltre, poiché viene utilizzata la classe lite, il modello deve essere ottimizzato prima di essere convertito. Ora siamo pronti per l'utilizzo del modello nel processo.

Capitolo 4

Risultati

In questo capitolo analizzeremo i risultati raggiunti durante la fase di testing dell'applicazione. Dopodiché gli stessi si andranno a comparare per trovare similitudini e differenze. Infine si cercherà di identificare il metodo migliore per raggiungere gli obiettivi più efficienti.

4.1 I test

Innanzitutto è necessario comprendere come sono stati realizzati i test. Partiamo dalle condizioni in cui si sono svolti: i test sono stati prodotti durante le ore centrali della giornata per assicurarsi che ci fosse abbastanza luce all'interno dell'auto, inoltre sono stati realizzati dieci test per il conducente e dieci per il passeggero. È importante sottolineare che, come anticipato nel paragrafo 3.3, verranno testate accuratamente tutte le possibili combinazioni dei parametri utilizzati per assicurarsi che non ci sia disparità tra i dati. I tempi utilizzati rappresentano una fase completa del frame, dalla sua acquisizione fino al termine della sua analisi da parte della Window.

È importante inoltre analizzare il modo in cui vengono calcolati i risultati. All'interno di una finestra si calcola la quantità di frame associata ad un determinato gruppo. Nel momento in cui questo valore supera la soglia Th, la finestra può essere considerata soddisfatta. La tipologia di finestra corrisponde al tipo di euristica scelta per ritenere un frame valido. Considerando G l'insieme di gruppi associati ad un frame si può dire che:

Size	1	3	5	8	10	20	30
Threshold	51%	60%	70%	80%			
Offset	0	5	10	30			
Type	*Single*	*Multiple*					

Tabella 4.1: Descrizione di tutti i possibili valori dei parametri del Window

Threshold	51%	60%	70%	80%

Tabella 4.2: Descrizione di tutti i possibili valori del parametro del Model

Single un frame è valido se $|G| = 1$, cioè se nell'immagine sono presenti solo oggetti di un solo gruppo.

Multiple un frame è valido se $|G| \geq 1$, cioè se nell'immagine è presente almeno un oggetto indipendentemente dal gruppo.

Pertanto il caso in cui il frame ha $|G| < 1$ non viene preso in considerazione da nessuna delle due tipologie.

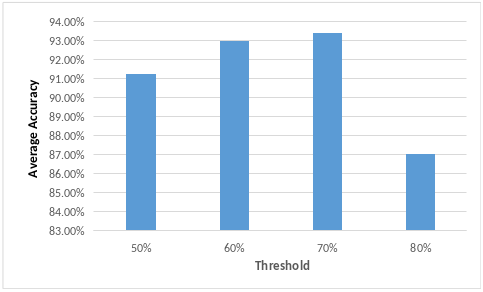
I parametri utilizzati per i vari test sono descritti nella tabella 4.2

Ora si analizzeranno i dati raccolti ed aggregati prima per valutazione e poi per finestra. Per ognuno di essi si osserveranno l'accuratezza, la confidence ed il tempo impiegato per concludere l'analisi. Infine si studierà come cambiano i dati aggiungendo un offset.

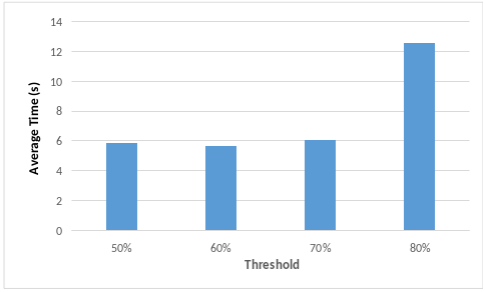
Per distinguere i dati si è deciso di utilizzare Acc, $Conf$, $Time$, rispettivamente per l'accuratezza, la confidence ed il tempo.

4.2 Valutazione

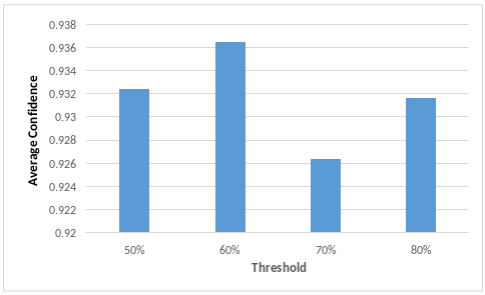
In questo paragrafo si analizzeranno i dati presenti nella figura 4.1, considerando unicamente i dati legati alle valutazioni.



(a) Rappresentazione dell'accuratezza



(b) Rappresentazione del tempo



(c) Rappresentazione della confidence

Figura 4.1: Analisi dei dati delle valutazioni nell'asse x si ha il threshold del modello, mentre nell'asse y si hanno i secondi o il valore percentuale del dato

Accuratezza

Le valutazioni, nel complesso, si può dire rispecchino accuratamente la realtà poiché, indipendentemente dal tipo di K utilizzato, si ottengono risultati superiori al 85%. Dal grafico della figura 4.1(a) si evince come il modello performi meglio con il threshold $K = 60\%$ e $K = 70$.

Confidence

Con riferimento alla confidence, si noterà che i dati possiedono in media, grossomodo, lo stesso valore, infatti tra il valore minimo e massimo c'è una differenza del 1%.

Tempo medio

La quantità media di tempo necessaria rimane pressoché la stessa, intorno ai 6s, a parte quella di $K_n = 80\%$ che è nettamente superiore, arrivando a totalizzare il doppio del tempo. Questo dimostra le difficoltà che il modello riscontra nelle analisi dei frame con un K eccessivamente alto, rendendo le analisi più difficili e, di conseguenza, più lunghe.

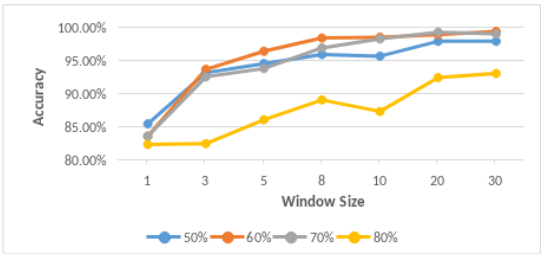
4.3 Window

I dati analizzati in questo paragrafo riguarderanno le Window. Verranno discussi due punti di vista: il primo riguardante K e il secondo Th, rispettivamente il threshold del modello e della finestra.

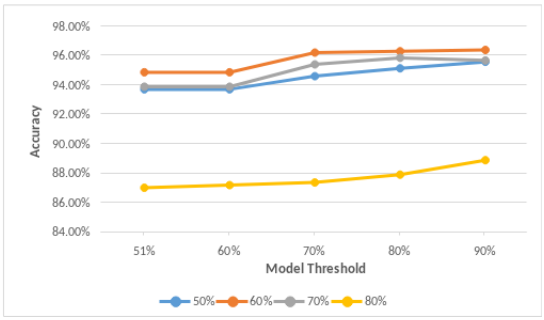
4.3.1 Accuratezza

Model Threshold

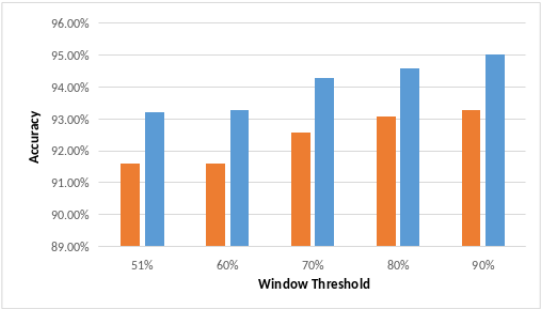
Nel grafico della figura 4.3 si può notare come l'accuratezza migliori con l'aumentare della lunghezza della finestra. Contrariamente a ciò che ci si poteva aspettare, i test peggiori, da questo punto di vista, sono quelli con il threshold K più alto. Infatti si può osservare come questi presentino delle percentuali nettamente inferiori rispetto agli altri.



(a) Rappresentazione dell'accuratezza



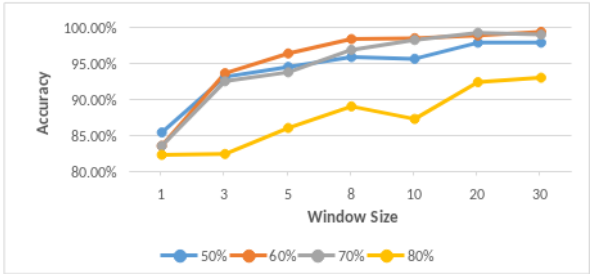
(b) Rappresentazione del tempo



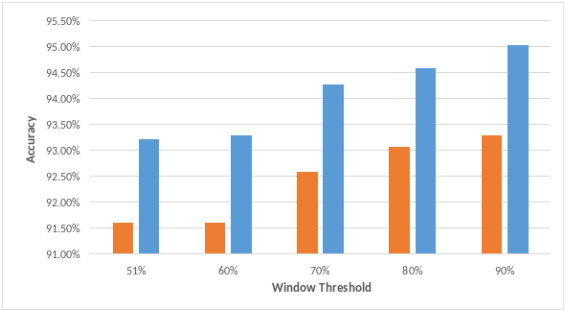
(c) Rappresentazione della confidence

Figura 4.2: Analisi dei dati delle valutazioni nell'asse x si ha il threshold del modello, mentre nell'asse y si hanno i secondi o il valore percentuale del dato. La legenda di (a) e (b) descrive i valori di K

Questo fenomeno è presente poiché il modello performa meglio in base alla quantità rispetto alla qualità. Dato che gli oggetti che il modello deve riconoscere sono molto simili, può capitare che l'analisi errata di un modello abbia una confidence particolarmente alta. Pertanto, data la difficoltà del modello a riconoscere oggetti con $K > 80\%$, gli errori commessi da esso avranno un'influenza maggiore nel complesso. Nei casi migliori i risultati prodotti sono intorno al 99%, che corrisponde ad un ottimo risultato. Il peggiore, cioè quello prodotto usando una finestra con $S = 1$, ha generato risultati tra l'82% e l'85%. Nel secondo grafico della figura 4.3 si ha la comparazione dei risultati utilizzando sia il threshold del modello, che quello della finestra. Si può notare facilmente come per ogni K si abbia un incremento in $Th = 70\%$. Inoltre è evidente come $K = 80\%$ sia nettamente inferiore come qualità rispetto agli altri tre dati, rispecchiando gli stessi valori descritti nella figura 4.1(a). Nell'ultimo grafico della figura 4.3 è descritta la differenza di accuratezza tra le due tipologie. L'andamento è ovviamente simile a quello riscontrato precedentemente avendo $K = 80\%$ come termine con i valori peggiori. Ma considerando le due tipologie ne deriva che $Ty =$ "$Single$" ottiene un'accuracy migliore per ogni K analizzato.



(a) Rappresentazione della lunghezza



(b) Rappresentazione della tipologia

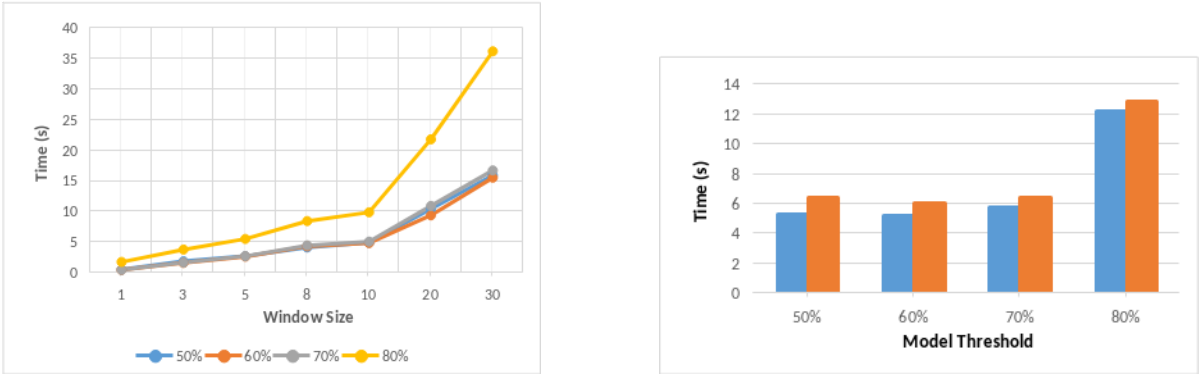
Figura 4.3: Analisi dei dati delle finestre nell'asse x si ha la lunghezza del Window oppure il suo threshold, mentre nell'asse y si ha la percentuale di accuratezza. La legenda di (a) descrive i valori di K

Window Threshold

Nella figura sono descritti due grafici, il primo che aggrega i dati in base alla lunghezza della finestra, il secondo in base alla tipologia. Nel primo grafico si può individuare un miglioramento dell'accuratezza con l'incremento della lunghezza, e i vari Th seguono pressoché lo stesso andamento. Nel caso della tipologia si nota un miglioramento dei risultati all'aumentare del Th e $Ty = "Single"$, che mantiene sempre la migliore accuratezza.

4.3.2 Tempi per completare una valutazione

Per il calcolo dei tempi medi si è deciso di utilizzare solamente i dati aventi l'offset $O = 0$, poiché l'applicazione dei diversi offset può portare a valori non validi. L'analisi in base all'offset sarà effettuata alla fine.



(a) Rappresentazione della lunghezza

(b) Rappresentazione della tipologia

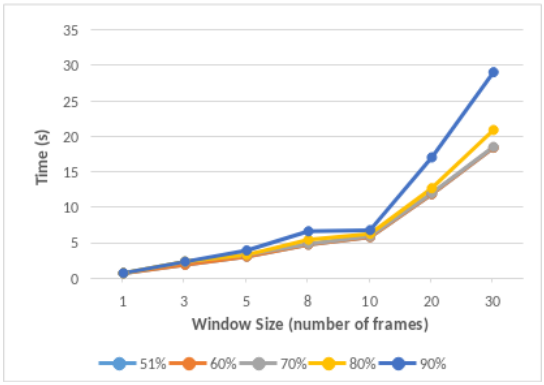
Figura 4.4: Analisi dei dati delle finestre nell'asse x si ha la lunghezza del Window oppure il threshold del modello, mentre nell'asse y si ha il tempo medio. La legenda in (a) descrive i valori di K

Model Threhsold

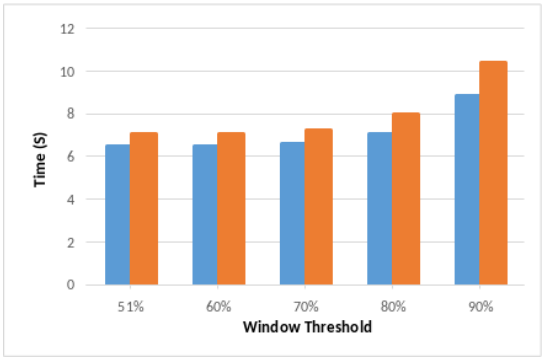
Partiamo dalle analisi riguardanti il threshold del modello. Ad ogni valore di K viene associato un parametro della Window tra lunghezza, threshold e tipologia.

La lunghezza gioca un fattore molto importante nella durata totale della valutazione quindi, come si può immaginare, si ha un incremento pari alla quantità di frame necessari a completare la finestra. Se si considera anche il K, si riscontra che per $K < 80\%$ si rilevano dei tempi pressoché simili, mentre all'aumentare di S cresce anche il distacco di $K = 80\%$. Lo stesso andamento si presenta dal punto di vista del Th e Ty. L'unico accorgimento in più lo si riconosce nella tipologia in cui $Ty =$ "$Multiple$" ha dei risultati sempre migliori rispetto a $Ty =$ "$Single$".

Come spiegato nel paragrafo 4.2 il motivo è dato dalla difficoltà del modello a riconoscere oggetti con $Conf > 80\%$.



(a) Rappresentazione della lunghezza

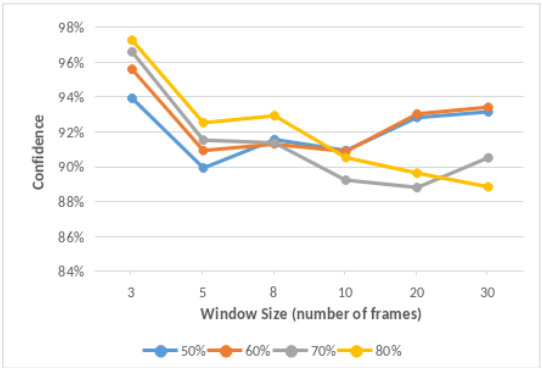


(b) Rappresentazione della tipologia

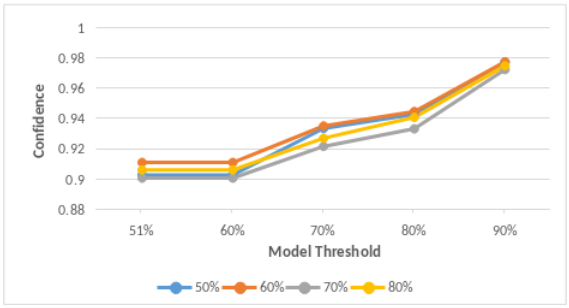
Figura 4.5: Analisi dei dati delle finestre nell'asse x si ha la lunghezza del Window oppure il il suo threshold, mentre nell'asse y si ha il tempo medi. La legenda in (a) descrive i valori di Th

Window Threshold

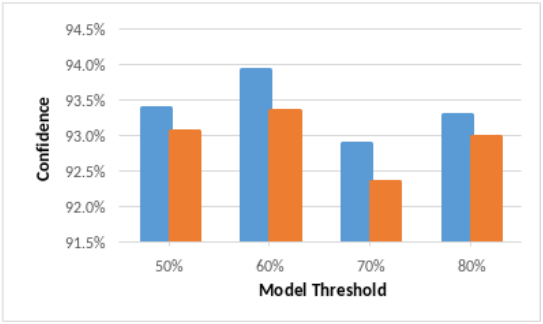
Ora useremo come base il threshold della finestre e lo compareremo con gli altri parametri del Window. Sono stati composti due grafici, uno per la lunghezza della finestra e l'altro per la tipologia. Entrambe le figure dimostrano come i dati tendano ad evidenziare la stessa quantità di tempo medio per completare il ciclo, differenziandosi solo nel caso $Th = 90\%$. In particolare, guardando Fig. 4.5(a) si nota che le differenze diventano più evidenti da $S > 10$.



(a) Rappresentazione della lunghezza



(b) Rappresentazione della soglia di Window



(c) Rappresentazione della tipologia

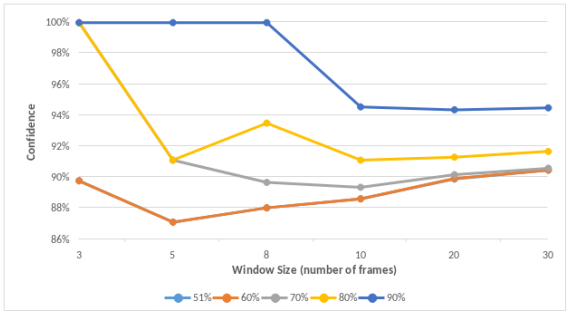
Figura 4.6: Analisi dei dati delle valutazioni nell'asse x si ha il Th, K o S, mentre nell'asse y si ha la percentuale di confidence. La legenda in (a) e (b) descrive i valori di K

4.3.3 Confidence

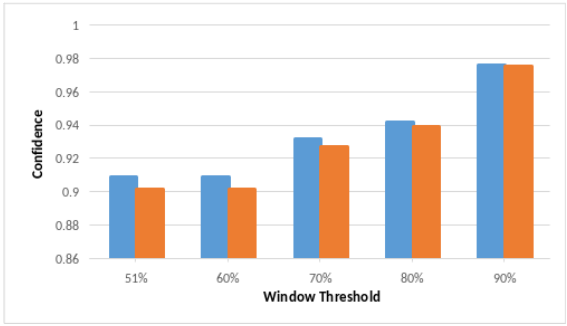
La confidence è il valore utilizzato per esprimere quanta fiducia si ha sulla correttezza del risultato. Il dato è molto interessante poiché non è legato alla realtà, è solo la probabilità che esso rispecchi la realtà. Questo è anche l'unico dato su cui l'applicazione basa i propri risultati. I risultati sono stati analizzati filtrando e suddividendo i dati con la stessa modalità spiegata nei due paragrafi precedenti.

Model Threshold

Prendendo in considerazione la lunghezza della finestra si potrà osservare nel grafico che i valori $K > 65\%$ hanno una maggiore confidence con $S < 10$, mentre la situazione si



(a) Rappresentazione della lunghezza



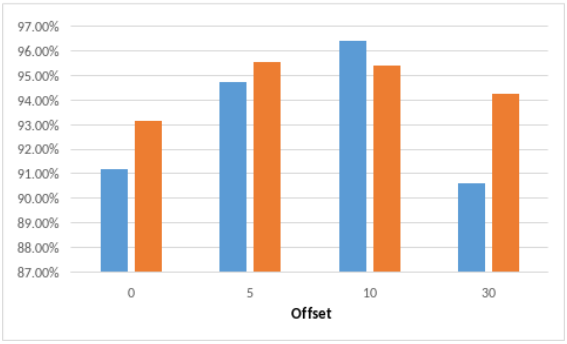
(b) Rappresentazione della tipologia

Figura 4.7: Analisi dei dati delle valutazioni nell'asse x si ha la lunghezza o threshold del Window, mentre nell'asse y si ha la percentuale di confidence. La legenda di (a) descrive i valori di Th

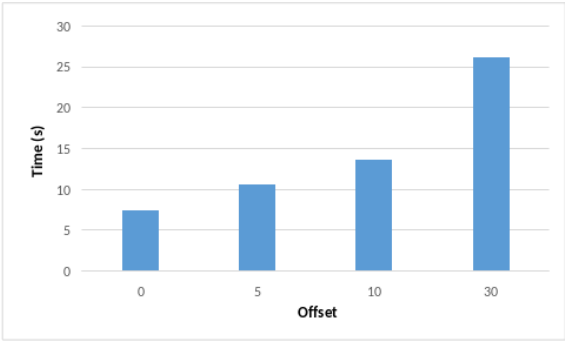
ribalta per $S \geq 10$. Se si prende esamina il parametro Th si osserva che, col crescere del threshold della finestra, cresce altrettanto la fiducia mentre non si nota una particolare differenza tra i valori K. L'ultimo grafico riguarda la tipologia, e qui si nota come i Multiple tendono ad avere una confidence migliore, anche se tutti i valori analizzati si assomigliano tra loro, infatti sono tra il 92% e il 94%.

Window Threshold

Il grafico 4.7(a) che rappresenta la lunghezza non fornisce molte informazioni poiché il calcolo della confidence è strettamente legato ai due parametri Th e S. Infatti con $S < 9 \wedge Th = 90\%$ si noterà una $Conf = 100\%$ che non è dovuta alla bontà del modello ma con un semplice calcolo si può comprendere che è l'unico risultato possibile con questa particolare combinazione. Infatti, prendendo come esempio $S = 8$, per far superare la soglia $Th = 90\%$ deve verificarsi che 5 immagini su 5 abbiano lo stesso gruppo. Pertanto si può constatare come questo particolare grafico non dia particolari informazioni utili se non che, nonostante $K_{=}51\%$, si abbia una confidence tra 88% e 90% che è molto buona. Osservando il secondo grafico di fig. 4.7 si noterà come *Multiple* anche in questo caso risulti con una confidence maggiore rispetto a *Single*, ma più grande è Th e più assottiglia la differenza arrivando quasi allo stesso valore in $Th = 90\%$.



(a) Rappresentazione della confidence e accuracy



(b) Rappresentazione del tempo medio

Figura 4.8: Analisi dei dati delle valutazioni nell'asse x si ha l'offset, mentre nell'asse y si hanno i secondi o l'accuratezza del dato

4.4 Utilizzo degli offset

Gli ultimi grafici da analizzare si riferiscono agli offset, cioè quanto si modifica il risultato se si attende una certa quantità di frame prima di rendere la finestra terminabile. I grafici studiati riguardano la fiducia, l'accuratezza ed il tempo medio. In questo caso la confidence e l'accuracy sono state aggiunte nello stesso grafico per semplificare la comparazione. Indagando se esiste qualche cambiamento nell'accuratezza della valutazione, si può osservare nella fig. 4.8(a) che i dati agli estremi sono relativamente simili, invece per $O = 5 \wedge O = 10$ migliorano di almeno il 5%. Lo stesso andamento lo si rileva per la confidence. Questo significa che aggiungendo un certo Δt gli input risultano più facilmente classificabili e accurati.

Osservando invece i tempi medi si nota come tendano ad aumentare di circa 3 sec ogni 5 frame, portando la valutazione con $O = 30$ ad avere una media di 26s.

4.5 Come stabilire il risultato migliore

Per concludere, andremo a determinare come ottenere i risultati migliori. Per poter arrivare a tale conclusione c'è bisogno di ricordare il contesto della tesi, infatti l'obiettivo

del progetto è quello di prevenire l'utilizzo del cellulare da parte di chi sta guidando un'automobile. In tal senso è necessario decidere alcune euristiche:

- una valutazione, prima di considerare il risultato valido, deve superare un certo limite Th_{MIN} di confidence che deve essere almeno del 50%;
- una valutazione deve avere un limite massimo di tempo $Time_{MAX}$ per completarsi. Al raggiungimento del limite si utilizza il valore calcolato anche se non tutte le Window son state soddisfatte;
- l'unico caso in cui una valutazione può superare $Time_{MAX}$ è se $Conf < Th_{MIN}$;
- porre un limite massimo Th_{MAX} di confidence per poter decretare il risultato con certezza;
- è necessario dare più importanza ad un falso positivo rispetto ad un falso negativo per poter mantenere la funzionalità di prevenzione.

Considerando queste euristiche, i risultati migliori non si otterranno utilizzando una sola combinazione di parametri, bensì usando un set di Window composto dalle finestre che danno l'accuratezza migliore senza compromettere la quantità di tempo necessaria per concludere. Infatti, come è stato detto antecedentemente, la confidence non è un valore certo, è solo la probabilità che il dato sia corretto. Come tale ci si deve aspettare che, nonostante una confidence particolarmente alta, il risultato non rispecchi la realtà. L'utilizzo di un set di Window, va a limitare nel maggior parte dei casi questi possibili errori di giudizio, senza impattare negativamente sul tempo di valutazione.

4.6 In sintesi

Dopo l'analisi dei dati e dei grafici, possiamo affermare che:

- aumentare Th non porta necessariamente a risultati migliori. Infatti nel nostro caso, con Th massimo si hanno tempistiche più lente, confidence e accuratezza minori. Il Th deve essere scelto in base alle prestazioni del modello. L'obiettivo infatti dovrebbe essere ottenere accuratezza migliore con tempi migliori;

- i valori di accuratezza nel caso peggiore sono superiori al 85% mentre nel caso migliore vanno ben oltre il 95%, dimostrando l'efficacia del modello in una situazione reale;
- si è inoltre osservato che, considerando l'accuratezza, si hanno i risultati migliori con $O = 10$ i quali tuttavia necessitano una quantità di tempo in più di circa 5/6s secondi;
- nonostante i vari grafici che determinano i migliori valori prendendo separatamente l'accuratezza e il tempo, nel momento in cui li si compara con gli obiettivi da noi richiesti si ottengono risultati differenti;
- la confidence può essere un buon termine per determinare se il risultato è corretto. Infatti, i pochi risultati non corretti ottenuti, tendono a presentare delle confidence più basse rispetto a quelli corretti;
- i risultati ottenuti da questo progetto rispecchiano i dati raccolti dalla tesi [11], ottenendo oltre tutto dei risultati migliori nell'accuratezza e nella confidence.

Conclusioni

In questo progetto abbiamo presentato una soluzione pratica al problema dell'uso del cellulare alla guida. L'applicazione verifica se, in una situazione reale, l'utilizzatore di uno smartphone in auto si trova alla guida o meno. L'app utilizza il modello YOLO che, tramite la tecnica *object detection*, identifica degli oggetti nelle immagine date in input. I frame vengono catturati dalla fotocamera frontale, i quali saranno poi utilizzati per classificare il frame. Abbiamo inoltre la possibilità di rappresentare i risultati del modello in tempo reale visualizzando quale oggetto è stato identificato, quale classe gli è stata associata e la *confidence*. Queste analisi sono possibili grazie all'utilizzo di finestre scorrevoli che analizzano solamente le ultime foto acquisite. Abbiamo anche dato la possibilità all'utente di specificare i parametri della Window e di poter comparare i risultati finali tra finestre con parametri differenti. L'app è stata sviluppata tenendo in considerazione anche la privacy dell'utente, infatti il modello e le analisi avvengono direttamente nel cellulare limitando quindi il passaggio di dati attraverso società terze. Questo permette anche di non doversi connettere ad internet per realizzare i test. L'applicazione dà la possibilità ad uno sviluppatore di testare i propri modelli cambiando poche righe di codice, e all'utente di visualizzare come esso funziona. Nonostante l'utilizzo di metodi diversi nella comparazione dei risultati rispetto alla tesi di Spallone [11], abbiamo ottenuto dei risultati riconducibili ad essa, infatti i livelli di *accuracy* e confidence nei casi medi sono nettamente al di sopra del 90%. Abbiamo inoltre definito il progetto affinché sia generale e relativamente semplice da estendere, grazie all'utilizzo di interfacce e classi astratte, che implementano le funzioni base, con riferimento al livello di astrazione interessato.

Ci sono molteplici estensioni del progetto che potranno essere sviluppate in futuro, un esempio può essere l'utilizzo di modelli diversi, sempre nell'ambito dell'*object detection*,

per poter comprendere qual è il modello migliore da impiegare. Altre possibilità sono quelle di inglobare il progetto di Saa applicando delle euristiche diverse nelle Window e riconoscere oggetti in più per poter distinguere anche se la persona si trova nelle postazioni anteriori o posteriori. Un'ulteriore possibilità futura sarà invece l'utilizzo di altri tipi di modelli come quello sviluppato da Octavian [10] per poi analizzare le prestazioni nel momento in cui vengono considerati i risultati di entrambi i modelli.

Infine proveremo a rispondere ad alcune domande che non sono state affrontate in questa tesi e che sono necessarie per la completezza dell'elaborato:

- come riconoscere se l'utente è all'interno di un'auto?
- quando si può iniziare ad analizzare le immagini della fotocamera? E quando è bene terminare?
- quali azioni compiere quando si riconosce che l'utilizzatore del device è il guidatore?

Bibliografia

[1] https://datareportal.com/reports/digital-2023-october-global-statshot

[2] https://www.sicurauto.it/news/codice-della-strada/codice-della-strada-2023-il-governo-lo-cambiera-cosi/

[3] https://images.squarespace-cdn.com/content/v1/5b79011d266c077298791201/4a02cbe6-cada-4af1-9f20-da9b5ce2004e/02+Internet+Time+QOQ+-+DataReportal+20231016+Digital+2023+October+Global+Statshot+Report+26.png?format=2

[4] https://www.addictioncenter.com/drugs/phone-addiction/

[5] https://www.brocardi.it/codice-della-strada/titolo-v/art173.html

[6] https://www.asaps.it/p/74216

[7] https://www.sicurauto.it/news/sistemi-di-sicurezza/distrazioni-al-volante-liihs-studia-nuove-soluzioni-basate-sulle-app/

[8] https://www.laleggepertutti.it/461820_uso-del-telefono-alla-guida-rischi-e-ritiro-di-patente

[9] Luca Bedogni, Marco Di Felice, Luciano Bononi, "Context-aware Android applications through transportation mode detection techniques", https://iris.unimore.it/retrieve/e31e124e-1525-987f-e053-3705fe0a095a/wcm.2702.pdf

[10] Luca Bedogni, Octavian Bujor, Marco Levorato, "Texting and Driving Recognition Exploiting Subsequent Turns Leveraging Smartphone Sensors", https://ieeexplore.ieee.org/document/8793032

[11] Federico Montori, Luca Bedogni, Spallone Marco, "Texting and Driving: rilevazione di utilizzo dello smartphone alla guida tramite Object Detection e Machine Learning.", https://amslaurea.unibo.it/26228/

[12] M. García-García, A. Caplier and M. Rombaut, "Driver Head Movements While Using a Smartphone in a Naturalistic Context", https://hal.science/hal-01538606/document

[13] https://dl.acm.org/doi/10.1109/TMC.2016.2539171

[14] https://en.wikipedia.org/wiki/Overfitting?lang=en

[15] https://docs.roboflow.com/

Ringraziamenti

Al termine di questo elaborato, mi è d'obbligo ringraziare tutte le persone che mi hanno sostenuto durante il mio percorso universitario e di scrittura della tesi che avete appena letto.

Per prima cosa, vorrei ringraziare il mio relatore Prof. Federico Montori, per i suoi preziosi consigli e per la sua disponibilità.

Ringrazio la mia famiglia che mi sostiene e mi ha sostenuto in tutti questi anni.

Ringrazio i miei amici bolognesi che mi hanno spronato a dare sempre il meglio.

Grazie ai miei amici di sempre, Gianmaria, Thomas ed Alberto, per avermi sostenuto con costanza e per essere sempre al mio fianco.

Ringrazio Alessandro, Guillermo, Erij per avermi aiutato a superare i momenti più difficili, senza di voi non avrei mai potuto arrivare a questo importante traguardo!

Ed infine ringrazio tutte le persone che mi hanno visto crescere in questi anni.

Grazie.