



Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Creazione modulo di riconoscimento oggetti in AR con uso di Neural Network

Relatore: *Daniela Micucci*

Co-relatore: *Michele Sesana*

Relazione della prova finale di:

Andrea Moscatelli

Matricola 833139

Anno Accademico 2019-2020

Ringraziamenti

Un ringraziamento speciale alla mia famiglia e alla mia fidanzata che hanno creduto in me. È grazie al loro sostegno e al loro incoraggiamento se oggi sono riuscito a raggiungere questo traguardo.

Ringrazio la mia relatrice e il mio co-relatore, Daniela Micucci e Michele Sesana, i quali mi hanno consigliato durante il periodo di stesura della tesi e grazie ai quali ho intrapreso questa esperienza al di fuori dell'università.

Ringrazio le persone che sono state e sono tutt'ora al mio fianco, colleghi e amici, grazie ai quali ho vissuto a pieno l'esperienza universitaria.

Infine, dedico questa tesi a me stesso, ai miei sacrifici, alla mia caparbia e alla mia tenacia che mi hanno permesso di arrivare fin qui.

Indice

CREAZIONE MODULO DI RICONOSCIMENTO OGGETTI IN AR CON USO DI NEURAL NETWORK

1. INTRODUZIONE	1
1.1. SCOPO DEL DOCUMENTO	1
1.2. TXT E-SOLUTIONS	1
1.3. WEAVR	1
1.4. IL PROGETTO DI TESI	2
1.4.1. Introduzione al progetto	2
1.4.2. Obiettivi	3
2. ANALISI	4
2.1. SOA AR FRAMEWORK	4
2.1.1. ARCore	4
2.1.2. Vuforia	6
2.1.3. Wikitude	9
2.1.4. Valutazione	12
2.2. OBJECT RECOGNITION OPEN SOURCE	18
2.2.1. Darknet	18
2.2.2. MediaPipe	19
2.2.3. TensorFlow	20
2.3. SCELTE PER LA PROGETTAZIONE	21
3. DIMOSTRATORE TECNOLOGICO	23
3.1. PROTOTIPO	23
3.2. TECHNOLOGY STACK	23
3.3. MODELLO	24
3.3.1. Dataset	26
3.3.2. Preparazione training	27
3.3.3. Training	30
3.3.4. Test e valutazione	31
4. INTEGRAZIONE E VALUTAZIONE	35
4.1. DATASET SAMSUNG GALAXY S2	35
4.2. VALUTAZIONE	42
4.3. INTEGRAZIONE WEAVR	43
5. CONCLUSIONI E SVILUPPI FUTURI	48
5.1. VALUTAZIONE	48
5.2. VALUTAZIONE PERSONALE	48
5.3. SVILUPPI FUTURI	48
GLOSSARIO	49
RIFERIMENTI BIBLIOGRAFICI	51

Elenco delle figure

Figura 1.1: Logo TXT e-solutions	1
Figura 1.2: Logo Pacelab WEAVR	1
Figura 2.1: Logo ARCore	4
Figura 2.2: Logo Vuforia	6
Figura 2.3: Architettura Vuforia	7
Figura 2.4: Immagine Ground Plane	8
Figura 2.5: Struttura VuMark	9
Figura 2.6: Logo Wikitude	9
Figura 2.7: Architettura Wikitude	10
Figura 2.8: Campione di immagini	15
Figura 2.9: Logo Darknet	18
Figura 2.10: Logo algoritmo YOLO	19
Figura 2.11: Logo MediaPipe	19
Figura 2.12: Architettura algoritmo SSD (Single Shot MultiBox Detector)	19
Figura 2.13: Logo TensorFlow	20
Figura 2.14: Architettura Fast R-CNN	20
Figura 2.15: Performance algoritmi sul COCO Dataset	22
Figura 3.1: Technology stack	23
Figura 3.2: Architetture supportate da Barracuda	24
Figura 3.3: Fasi dell'esecuzione di YOLO	25
Figura 3.4: Rappresentazione dell'architettura di YOLO	26
Figura 3.5: Esempio di creazione di un'annotazione con il software LabelImg.	27
Figura 3.6: Logo Google Colaboratory	27
Figura 3.7: Parte del file di configurazione	28
Figura 3.8: Architettura della rete Darknet-19	29
Figura 3.9: Tempi preparazione dataset prototipo	29
Figura 3.10: Esempio di batch	30
Figura 3.11: Rappresentazione di Precision, Recall e della metrica IOU.	30
Figura 3.12: Tempi preparazione e training dataset prototipo	31
Figura 3.13: Risultato valutazione di "yolov2-tiny_last.weights"	31
Figura 3.14: Rappresentazione IoU	32
Figura 3.15: Esempio con previsioni corrette Figura 3.16: Esempio con previsioni corrette	33
Figura 3.17: Esempio con previsioni non corrette Figura 3.18: Esempio con previsioni non corrette	33
Figura 3.19: Numero immagini minime consigliate	34
Figura 4.1: Costi preparazione dataset	35
Figura 4.2: Punteggi dataset	37
Figura 4.3: Medie punteggi dataset	37
Figura 4.4: Dati ottenuti dai modelli con dataset differenti	38
Figura 4.5: Medie punteggi dataset (Dataset 3000 aggiornato)	38
Figura 4.6: Tempo in percentuale per ogni fase	39
Figura 4.7: Tempo training e totale per la creazione del dataset	39
Figura 4.8: Riconoscimento fronte (50) Figura 4.9: Riconoscimento retro (50)	40
Figura 4.10: Riconoscimento retro aperto (50)	40
Figura 4.11: Riconoscimento fronte (200) Figura 4.12: Riconoscimento retro (200)	40
Figura 4.13: Riconoscimento retro aperto (200)	41
Figura 4.14: Riconoscimento fronte (3000) Figura 4.15: Riconoscimento retro (3000)	41
Figura 4.16: Riconoscimento retro aperto (3000)	41
Figura 4.17: Struttura procedura	44
Figura 4.18: Esempio flow della procedura	44

Figura 4.19: Esempio del coding della procedura.....	45
Figura 4.20: Benvenuto della procedura.....	45
Figura 4.21: Rotazione del telefono	46
Figura 4.22: Riconoscimento fronte in AR.....	46
Figura 4.23: Aprire la cover	46
Figura 4.24: Riconoscimento fessura apertura cover AR	47
Figura 4.25: Rimuovere la batteria	47
Figura 4.26: Fessura per rimuovere la batteria AR	47

Elenco delle tabelle

Tabella 2.1: SDK e tecnologie supportate (*Wikitude è stato testato solo su webcam).....	12
Tabella 2.2: Legenda.....	12
Tabella 2.3: Oggetti utilizzati	12
Tabella 2.4: Confronto condizioni di luce su oggetto 1.	13
Tabella 2.5: Confronto condizioni di luce su oggetto 3.	13
Tabella 2.6: Confronto dimensioni differenti	13
Tabella 2.7: Confronto forme differenti	13
Tabella 2.8: Confronto multicolore e monocolori.....	14
Tabella 2.9: Piani riconosciuti	14
Tabella 2.10: Confronti riconoscimento piani con condizioni di luce differenti	15
Tabella 2.11: Confronto condizioni di luce differenti su immagine 2.....	15
Tabella 2.12: Confronto condizioni di luce differenti su immagine 1.....	15
Tabella 2.13: Confronto dimensione su immagine 4.	16
Tabella 2.14: Confronto dimensione su immagine 6.	16
Tabella 2.15: Confronto monocolori e multicolori	16
Tabella 2.16: Confronto forma su immagini multicolori	16
Tabella 4.1: Esempio valutazione modello.....	36
Tabella 4.2: Legenda per punteggi	42
Tabella 4.3: Valutazione alto livello.....	43
Tabella 4.4: Valutazione numerica alto livello	43

1. Introduzione

1.1. Scopo del documento

Questo elaborato rappresenta la relazione finale riguardo al lavoro svolto dal sottoscritto durante il periodo di stage presso l'azienda TXT e-solutions come conclusione del percorso di laurea triennale. Lo scopo è descrivere dettagliatamente le mansioni svolte, dimostrando di aver compreso e messo in atto tutte le competenze acquisite durante questo corso di studi e integrate da una formazione specifica aziendale.

1.2. TXT e-solutions



Figura 1.1: Logo TXT e-solutions

TXT e-solutions è un fornitore internazionale specializzato di soluzioni software di ingegneria che supporta i clienti nei mercati *high-tech* nei loro processi *core mission-critical* e *business* e durante tutto il ciclo di vita del prodotto. Fondata nel 1989, la società è quotata alla Borsa Italiana - segmento STAR (TXT.MI) - dal luglio 2000 e conta circa 1.000 dipendenti. TXT e-solutions ha sede a Milano e ha filiali in Italia, Germania, Regno Unito, Francia, Svizzera e Stati Uniti.

1.3. WEAVR



Figura 1.2: Logo Pacelab WEAVR

Pacelab WEAVR è una *software suite* che semplifica la creazione, la distribuzione e l'implementazione di soluzioni di realtà aumentata, virtuale e mista su applicazioni cloud e PC, VR e dispositivi mobili. È rivolto a professionisti dell'ingegneria nel settore aerospaziale e aeronautico. Può essere utilizzato per configurazioni di prodotti, formazione e simulazione, attività di assemblaggio e manutenzione.

La piattaforma Pacelab WEAVR è composta da tre moduli che insieme forniscono tutto il necessario per implementare e gestire la formazione virtuale:

- WEAVR Creator è lo strumento di creazione intuitivo e senza necessità di scrivere codice per la creazione dei contenuti di formazione.

- WEAVR Player, app rivolta agli studenti per un facile accesso ed esecuzione della formazione in *cross reality* (XR).
- WEAVR Manager gestisce gli utenti, i contenuti, la collaborazione *multi-player*, l'analisi dei risultati e molto altro.

1.4. Il progetto di tesi

1.4.1. Introduzione al progetto

Il progetto è stato svolto durante il periodo di stage con una durata di 6 mesi, dal 06/07/2020 al 22/01/2021, principalmente in modalità *smart working* e, talvolta, presso la sede milanese di TXT e-solutions S.p.A. in Via Frigia 27.

Il progetto in questione ha come obiettivo principale la ricerca e la creazione di un modulo per effettuare l'*object recognition* in *Augmented Reality* (AR) usando una *Neural Network* (NN) per la possibile integrazione di questa tecnologia nella piattaforma WEAVR sviluppata da TXT per sostituire le librerie attuali di Vuforia. Nello specifico, inizialmente è stato creato un prototipo, il quale è stato successivamente testato su un modello in grado di riconoscere il telefono "Samsung Galaxy S2". Infine, è stata implementata una procedura per il supporto operativo (OpS) di sostituzione della batteria del telefono.

Nel corso di tale attività, è stato approfondito il mondo dell'*Augmented Reality* nello studio e nello sviluppo di applicazioni per PC e Android. Il mercato della *augmented reality* ha una base di installazioni attiva che si è avvicinata a 900 milioni e oltre 8 miliardi di dollari di entrate nel 2019 [1]. Si prevede che i download totali dei consumatori di applicazioni di realtà aumentata in tutto il mondo nel 2022 supereranno i 5,5 miliardi [2].

L'*Augmented Reality* è la realtà così come percepita sensorialmente e intellettualmente dall'individuo, arricchita di dati in formato digitale. In sostanza, un potenziamento – mediante dispositivi ad alta tecnologia – delle capacità fornite dai cinque sensi e dall'intelletto. Non si tratta quindi di un mondo virtuale, bensì di un'integrazione fra realtà fisica e mondo digitale attraverso dispositivi comuni, come gli smartphone, oppure più sofisticati, come gli smart glasses.

Un esempio di *smart glasses* sono le Microsoft HoloLens che utilizzano il sistema operativo Windows Mixed Reality. Si presentano come occhiali trasparenti che lasciano aperto il campo visivo dell'utente e totale libertà di movimento, permettendo di posizionare virtualmente finestre di app e immagini 3D su ciò che in quel momento viene focalizzato con lo sguardo [3].

Inoltre, è stato esplorato il mondo delle *Neural Networks*. Una *Neural Network* è un modello matematico composto da neuroni artificiali di ispirazione alle reti neurali biologiche (quella umana o animale) e viene utilizzata per risolvere problemi ingegneristici di Intelligenza Artificiale legati a diversi ambiti tecnologici come l'informatica, l'elettronica o altre discipline.

Più precisamente, le *neural networks* si possono definire come modelli di calcolo matematico-informatici basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni; tali interconnessioni derivano da neuroni artificiali e processi di calcolo basati sul modello delle scienze cognitive chiamato "connessionismo" [4].

Secondo un rapporto pubblicato da Allied Market Research, il mercato globale delle *neural networks* è stato valutato a \$7,03 miliardi nel 2016 e si stima che raggiungerà \$38,71 miliardi entro il 2023 [5].

Inizialmente, è stato studiato lo *State of Art* (SoA) di vari framework e software e la loro sperimentazione, come ad esempio ARCore, Vuforia, Wikitude, MediaPipe, TensorFlow e Darknet. Successivamente, è stato implementato un prototipo al fine di essere un dimostratore di questa tecnologia.

Infine, per il modulo finale, il prototipo è stato integrato e implementato in Unity usando un modello creato tramite il framework Darknet, l'algoritmo YOLO e due *packages* principali:

- Barracuda, una libreria di inferenza per reti neurali multiplatforma.
- WEAVR, come descritto in precedenza, una *software suite* sviluppata da TXT.

Tramite il WEAVR si è poi creata una procedura OpS per simulare una possibile implementazione. Essa è utilizzata come supporto per eseguire varie operazioni sul campo (ad es. Assistenza tecnica per addetti alla manutenzione). Solitamente, non consente la navigazione dell'utente nei mondi virtuali, bensì di interagire solo con gli elementi dell'interfaccia utente (UI).

You Only Look Once (YOLO) [6] è un algoritmo di *Deep Learning* per il rilevamento di oggetti. YOLO esegue il rilevamento degli oggetti classificandoli all'interno dell'immagine e determinando dove si trovano su di essa. Descritto per la prima volta nel seminale del 2015 di Joseph Redmon, YOLO è una delle implementazioni *Open Source Neural Network* framework, Darknet.

1.4.2. Obiettivi

Obiettivi personali

Gli obiettivi prefissati inizialmente sono stati:

- Studio dello *state of art* dei principali framework e software in AR.
- Acquisizione delle competenze nello sviluppo e integrazione del codice.
- Progettazione di un applicativo per effettuare l'*object recognition*.
- Collaborazione con colleghi aventi esperienza nell'ambiente di sviluppo.
- Approfondimento delle conoscenze nell'ambito dell'*Augmented Reality*.

Obiettivi dell'azienda

Gli obiettivi dell'azienda per questo progetto sono stati:

- Ricerca e sviluppo di nuove tecnologie.
- Sostituzione delle librerie di Vuforia per l'*object recognition*.
- Aumento del livello di personalizzazione del prodotto WEAVR.
- Formazione.

2. Analisi

2.1. SoA AR framework

Lo scopo di questa analisi è conoscere lo *State of Art*, approfondire e confrontare vari framework in *augmented reality*. È stato opportuno valutare quali tecnologie offrissero i framework e la loro efficacia.

La crescente popolarità dell'AR nello sviluppo di applicazioni può essere attribuita anche ai due maggiori colossi della tecnologia, Apple e Google, che hanno sviluppato i propri toolkit di sviluppo AR con framework e SDK per includere rispettivamente ARKit e ARCore.

Sono stati ricercati i framework più popolari e aventi una valutazione migliore. Secondo l'azienda di consulenze "8allocate", i primi 4 framework, nonché gli unici con una valutazione positiva, sono: ARKit, ARCore, Vuforia, Wikitude. [7]

Da questa valutazione è stato rimosso ARKit in quanto esso permette lo sviluppo solo per dispositivi iOS e, di conseguenza, data la poca disponibilità di strumenti anche per via dello *smart working*, non è stato possibile effettuare una pari comparazione. Inoltre, permettere di sviluppare soluzioni solo per dispositivi iOS costituisce già di per sé una limitazione importante.

Di conseguenza, si è deciso di mettere a confronto ARCore, Vuforia e Wikitude. In particolare, sono state studiate le funzionalità di *Object Recognition*, *Plane Tracking* e il riconoscimento di *QR code*.

2.1.1. ARCore



Figura 2.1: Logo ARCore

ARCore [8] è la piattaforma di Google per la creazione di esperienze di realtà aumentata. Utilizzando diverse API, ARCore consente al telefono di percepire il suo ambiente, comprendere il mondo e interagire con le informazioni. Alcune delle API sono disponibili su Android e iOS per abilitare esperienze AR condivise. ARCore utilizza tre funzionalità chiave per integrare i contenuti virtuali con il mondo reale visto attraverso la fotocamera del telefono:

- il rilevamento del movimento, il quale consente al telefono di comprendere e monitorare la sua posizione rispetto al mondo;
- la comprensione ambientale, che consente al telefono di rilevare le dimensioni e la posizione di tutti i tipi di superfici (superfici orizzontali, verticali e angolate come il pavimento, un tavolino da caffè o le pareti);
- la stima della luce, la quale consente al telefono di stimare le condizioni di illuminazione attuali dell'ambiente.

ARCore offre molteplici funzionalità: *points*, *plane detection*, *pose*, *light estimation*, *anchors*, *image tracking*, *face tracking*, *object occlusion*, e *cloud anchors*. Queste funzionalità sono simili ad ARKit.

Con le funzionalità di rilevamento della posizione in tempo reale e l'integrazione di oggetti virtuali e reali, ARCore è un ottimo strumento per AR nelle app di *e-commerce*. Ad esempio, i venditori su Ebay possono utilizzare AR per scegliere la grandezza giusta dei pacchi da utilizzare.

In ARCore 1.21.0, purtroppo, non è ancora stata implementata la funzionalità per effettuare l'*object recognition*. Quindi, solo le funzionalità di *Plane Tracking* e riconoscimento di *QR code* sono state testate con dei mini-progetti.

Plane Detection

ARCore utilizza una classe denominata *DetectedPlane* per rappresentare i piani rilevati. Al fine di rilevare i piani, si basa su due sue principali caratteristiche: *environmental understanding* e *depth understanding* [9].

- *Environmental understanding*: ARCore migliora costantemente la sua comprensione dell'ambiente del mondo reale rilevando punti e piani caratteristici. ARCore cerca gruppi di punti caratteristici che sembrano giacere su superfici orizzontali o verticali comuni, come tavoli o muri, e rende queste superfici disponibili all'applicazione come piani. ARCore è anche in grado di determinare il confine di ogni piano e rendere disponibili tali informazioni all'applicazione. È possibile utilizzare queste informazioni per posizionare oggetti virtuali appoggiati su superfici piane. Poiché ARCore utilizza i punti funzione per rilevare i piani, le superfici piatte senza consistenza, come ad esempio un muro bianco, potrebbero non essere rilevate correttamente.
- *Depth understanding*: ARCore può creare mappe di profondità, ovvero immagini contenenti dati sulla distanza tra le superfici da un dato punto, utilizzando la fotocamera RGB principale da un dispositivo supportato. È possibile utilizzare le informazioni fornite da una mappa di profondità per abilitare esperienze utente coinvolgenti e realistiche, come far collidere accuratamente oggetti virtuali con superfici osservate o farli apparire davanti o dietro oggetti del mondo reale.

QR Code

Attraverso l'Image Tracking è possibile tracciare e riconoscere *QR Code*. Le immagini aumentate in ARCore consentono di creare app AR in grado di rispondere alle immagini 2D nell'ambiente dell'utente, come poster o confezioni di prodotti. *Augmented Image* [10] è una funzionalità che consente di creare app AR in grado di rispondere a specifiche immagini 2D come la confezione del prodotto, i poster dei film o, in questo caso, dei *QR Code*. Gli utenti possono attivare esperienze AR quando puntano la fotocamera del telefono su immagini specifiche, ad esempio verso un poster di un film e far apparire un personaggio che recita una scena.

L'API *Augmented Image* presenta le seguenti funzionalità:

- ARCore è in grado di rispondere e tenere traccia delle immagini fisse, ad esempio una stampa appesa a una parete o una rivista su un tavolo. A partire da ARCore 1.9, il framework tiene traccia anche di immagini in movimento come una pubblicità su un autobus in transito o un'immagine su un oggetto piatto tenuto dall'utente mentre muove le mani.
- Dopo che ARCore inizia a tracciare un'immagine, fornisce stime per posizione, orientamento e dimensioni fisiche. Queste stime vengono continuamente perfezionate man mano che ARCore raccoglie più dati. Fornire una dimensione fisica accurata migliora i tempi di rilevamento delle immagini. Se non viene specificata alcuna dimensione fisica, ARCore stima la dimensione.
- Una volta che un'immagine è stata rilevata, ARCore è in grado di continuare a tracciare la sua posizione e il suo orientamento, anche se essa si è temporaneamente spostata fuori dalla vista della telecamera.

ARCore è in grado di tener traccia di un massimo di 20 immagini contemporaneamente nell'ambiente dell'utente *off-camera* e *on-camera*, ma non di più istanze della stessa immagine.

Tutto il monitoraggio avviene sul dispositivo, quindi non è necessaria alcuna connessione a Internet. Le immagini di riferimento possono essere aggiornate sul dispositivo o sulla rete senza richiedere un aggiornamento dell'app.

Le immagini possono essere compilate offline per creare un database di immagini oppure è possibile aggiungerle in tempo reale dal dispositivo. Una volta registrato, ARCore rileverà queste immagini, i loro confini e restituirà una posa corrispondente. ARCore può memorizzare informazioni sui punti per un massimo di 1.000 immagini di riferimento per database di immagini. Non c'è limite al numero di database, tuttavia può essere attivo un solo database in qualsiasi momento.

2.1.2. Vuforia



Figura 2.2: Logo Vuforia

Vuforia [11] è un kit di sviluppo software di *augmented reality* (SDK) per dispositivi mobili che consente la creazione di applicazioni in *augmented reality*. Utilizza la tecnologia di visione artificiale per riconoscere e tracciare immagini planari e oggetti 3D in tempo reale. Questa capacità di registrazione delle immagini consente agli sviluppatori di posizionare e orientare oggetti virtuali, come modelli 3D e altri supporti, in relazione agli oggetti del mondo reale quando vengono visualizzati attraverso la fotocamera di un dispositivo mobile. L'oggetto virtuale, quindi, traccia la posizione e l'orientamento dell'immagine in tempo reale in modo che la prospettiva dello spettatore sull'oggetto corrisponda alla prospettiva sul bersaglio. Di conseguenza, sembra che l'oggetto virtuale faccia parte della scena del mondo reale.

Vuforia supporta una varietà di tipi di target 2D e 3D, inclusi i target di immagine “*marker less*”, il 3D Model Target e una forma di Fiducial Marker indirizzabile, noto come VuMark. Le funzionalità aggiuntive dell'SDK includono la localizzazione del dispositivo a sei gradi di libertà nello spazio, il rilevamento dell'occlusione localizzato mediante “pulsanti virtuali”, la selezione del target dell'immagine a *run-time* e la capacità di creare e riconfigurare i set di target in modo programmatico in fase di *run-time*.

Vuforia Engine [12] è la piattaforma più utilizzata per lo sviluppo AR con supporto per i principali telefoni, tablet e occhiali. Gli sviluppatori possono aggiungere facilmente funzionalità avanzate di visione artificiale alle app Android, iOS e UWP per creare esperienze AR che interagiscono realisticamente con gli oggetti e l'ambiente. Vuforia supporta le tecnologie di *object recognition*, *plane tracking* e il riconoscimento di *QR code*.

Nella [Figura 2.3](#) si può osservare l'architettura di Vuforia e il relativo database.

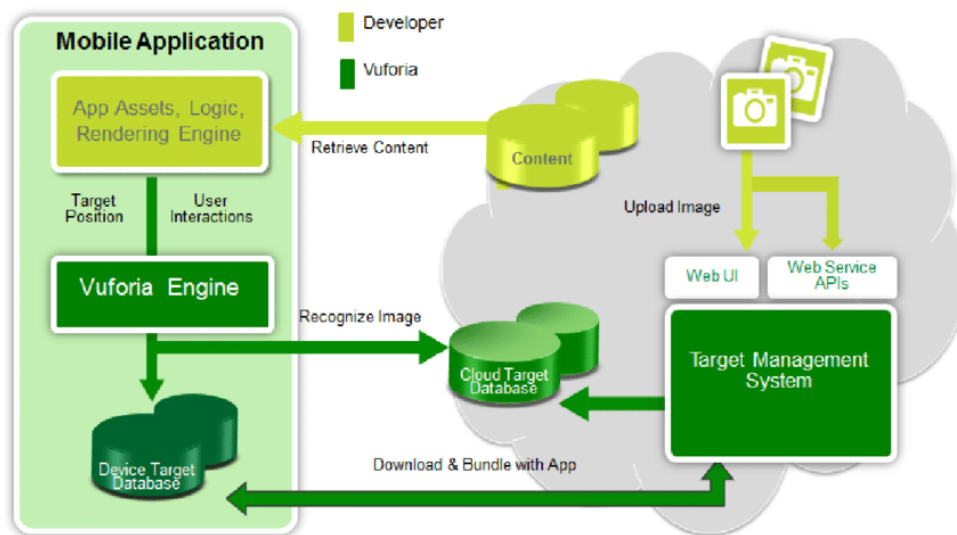


Figura 2.3: Architettura Vuforia

Object Recognition

Vuforia attraverso i Model Targets [13] è in grado di effettuare l'*object recognition*. I Model Targets consentono alle applicazioni create utilizzando Vuforia Engine di riconoscere e tracciare oggetti particolari nel mondo reale in base alla loro forma. Per creare un Model Target per un oggetto particolare, è necessario avere accesso ai dati del modello 3D, ad esempio un modello CAD 3D o una scansione 3D dell'oggetto creato o ottenuto da una fonte o applicazione di terze parti.

Vuforia mette a disposizione una sua applicazione per avere accesso ai dati di qualunque oggetto in modo semplice e veloce: Vuforia Object Scanner [14].

Vuforia Object Scanner è un'applicazione android utilizzata per scansionare un oggetto fisico 3D. Object Scanner produce un file Object Data (*.OD) che include i dati di origine richiesti per definire un Object Target in Target Manager. Lo Scanner consente di generare, testare e modificare file (*.OD). Inoltre, fornisce una visualizzazione delle caratteristiche dell'oggetto e della loro copertura su di esso. Vuforia mette a disposizione un'altra applicazione per il riconoscimento degli oggetti in maniera più professionale: Model Target Generator [15]. Il Model Target Generator prende come input un modello 3D che rappresenta l'oggetto che si desidera monitorare, ne verifica l'idoneità, consente di impostare gli angoli e la distanza della vista guida e, infine, genera un database Vuforia che è possibile utilizzare con l'integrazione di Unity di Vuforia Engine o nell'applicazione nativa al fine di monitorare l'oggetto.

Plane Tacking

Vuforia offre due possibilità: Ground Plane e Mid-Air.

Vuforia Ground Plane [16] consente di posizionare i contenuti digitali su superfici orizzontali nell'ambiente, come pavimenti e tavoli. Supporta il rilevamento e il tracciamento di superfici orizzontali e consente anche di posizionare il contenuto a mezz'aria utilizzando gli Anchor Points. Ground Plane utilizza l'API Vuforia SmartTerrain insieme a Device Tracking.

Ground Plane si basa su dettagli visivi nell'ambiente per rilevare e tracciare i piani, nonché sulla posizione dell'utente nel mondo. Un indicatore di superficie è un prezioso strumento di interfaccia utente per informare gli utenti su dove e quando possono posizionare i contenuti nelle loro impostazioni. I campioni del Ground Plane Vuforia dimostrano l'uso di un indicatore di superficie e mostrano come modificarne la presentazione per riflettere le condizioni della superficie. Ha bisogno quindi di un

elemento fisico che rappresenti il piano d'appoggio per iniziare a rilevare il piano vero e proprio. Di seguito, nella Figura 2.4, viene mostrato l'elemento fisico che rappresenta il piano d'appoggio.



Figura 2.4: Immagine Ground Plane

Un'estensione del Ground Plane è il posizionamento a Mid-Air [16]. Il contenuto può essere collocato in una posizione definita rispetto al dispositivo. Questo procedimento viene eseguito a livello di programmazione e richiede che lo spazio globale venga precedentemente inizializzato. I campioni di Ground Plane forniscono un'interfaccia utente per il posizionamento a Mid-Air che utilizza una metafora a mirino. Il design dell'interfaccia utente del posizionamento a mezz'aria può essere personalizzato per potersi adattare a una varietà di casi d'uso.

QR Code

Vuforia per la lettura dei *QR code* mette a disposizione due possibilità.

La prima, nel modo classico, tramite l'Image Target [17]. Image Targets rappresentano immagini che Vuforia Engine è in grado di rilevare e tracciare. L'Engine rileva e tiene traccia dell'immagine confrontando le caratteristiche naturali estratte da quelle della telecamera con un database di risorse di destinazione noto. Una volta rilevato l'Image Target, Vuforia Engine monitorerà l'immagine e aumenterà i contenuti senza soluzione di continuità, utilizzando la migliore tecnologia di tracciamento delle immagini sul mercato. Le funzionalità estratte da queste immagini sono archiviate in un database cloud o dispositivo, il quale è possibile scaricare e impacchettare insieme all'applicazione. Il database può quindi essere utilizzato da Vuforia Engine per confronti di *run-time*. Un'altra opzione è quella di creare target di immagini all'interno di Unity Editor utilizzando direttamente gli asset di immagini o utilizzando l'API corrispondente in fase di esecuzione.

La seconda possibilità è quella di utilizzare il VuMark. VuMark [18] è il codice a barre di prossima generazione, il quale offre la libertà per un design personalizzato e attento al marchio, codificando contemporaneamente i dati e fungendo da target AR. I design VuMark sono completamente personalizzabili, quindi si può avere un VuMark unico per ogni oggetto. Lo stesso design VuMark può essere utilizzato per codificare una serie di ID o dati univoci. Ciò è particolarmente utile quando è necessario essere in grado di utilizzare la stessa immagine universalmente, ma avente un'identità e / o informazioni univoche.

Nella [Figura 2.5](#) si possono osservare i vari componenti di un generico VuMark.

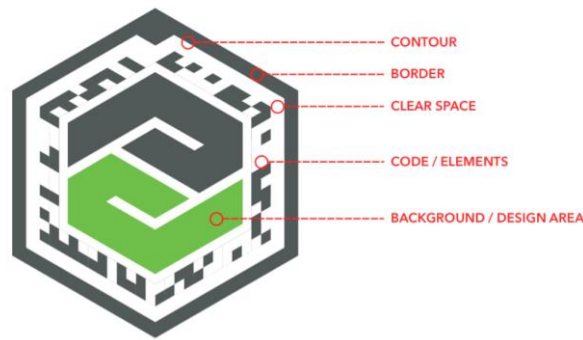


Figura 2.5: Struttura VuMark

2.1.3. Wikitude



Figura 2.6: Logo Wikitude

Wikitude è un fornitore di tecnologia di *augmented reality mobile* con sede a Salisburgo, Austria. Fondato nel 2008, Wikitude si è inizialmente concentrato sulla fornitura di esperienze di realtà aumentata basate sulla posizione attraverso l'applicazione Wikitude World Browser. Nel 2012, la società ha ristrutturato la sua proposta lanciando Wikitude SDK, un framework di sviluppo che utilizza il riconoscimento e il tracciamento delle immagini e le tecnologie di geolocalizzazione.

Nel 2017, Wikitude ha lanciato la sua tecnologia SLAM che consente il riconoscimento e il tracciamento degli oggetti, nonché il tracciamento istantaneo senza marker. L'SDK multiplatforma è disponibile per i sistemi operativi Android, iOS e Windows ed è ottimizzata anche per diversi dispositivi *smart eyewear*. Wikitude Native SDK è una libreria software e un framework per *app mobile* utilizzato per creare esperienze di realtà aumentata. L'SDK nativo supporta casi d'uso che richiedono il riconoscimento di immagini e la tecnologia di tracciamento (realtà aumentata basata sulla visione).

La [Figura 2.7](#) mostra l'architettura di Wikitude.

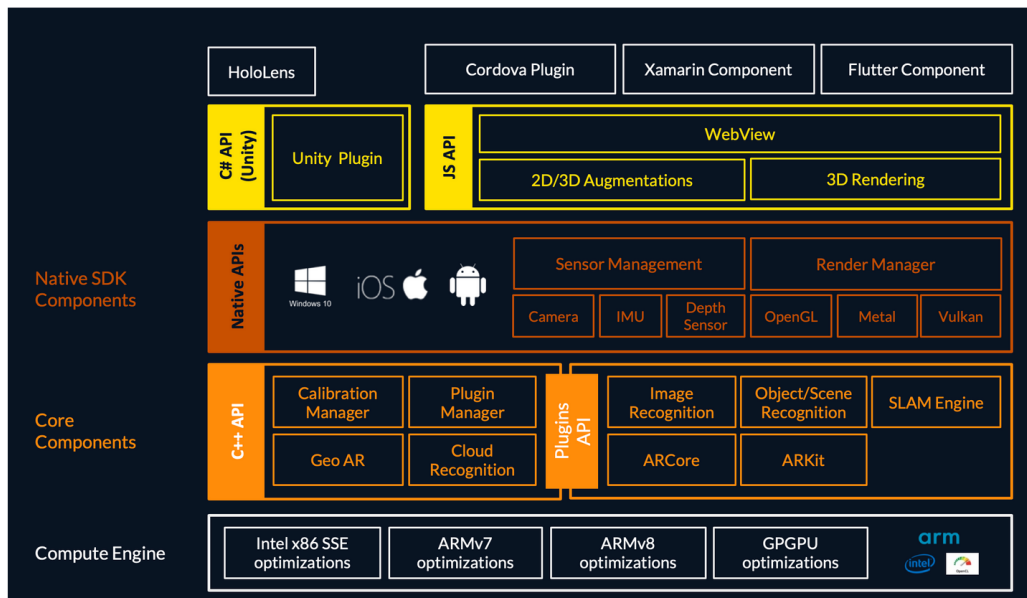


Figura 2.7: Architettura Wikitude

Object Recognition

L'*object recognition* [19] si suddivide in tre aree: Simple, Extended Tracking for Objects e Scene Recognition.

La funzione si basa sul motore SLAM di Wikitude utilizzato in tutto l'SDK per qualsiasi tipo di tracciamento dell'ambiente. Per tracciare un oggetto serve il Prefab ObjectTracker, il quale viene importato tramite SDK.

Per il Simple è necessario un ObjectTracker. Esso necessita di una raccolta di oggetti Wikitude (file “*.wto”) che contiene le informazioni necessarie per rilevare gli oggetti di riferimento. Le raccolte target possono essere generate e scaricate da Wikitude Studio, uno strumento web gratuito a cui si può accedere con il proprio account sviluppatore. È possibile utilizzare il file “*.wto” allo stesso modo dei file “*.wtc” utilizzati per Image Tracker.

L'Extended Tracking è una modalità opzionale che è possibile impostare separatamente per ciascun obiettivo. In questa modalità, Wikitude SDK proverà a continuare a scansionare l'ambiente dell'utente anche se l'oggetto target originale non è più in vista. Quindi il tracciamento si estende oltre i limiti dell'oggetto target originale. Le prestazioni di questa funzione dipendono da vari fattori come la potenza di calcolo del dispositivo, la trama di sfondo e gli oggetti.

Con la versione SDK 8, il motore di riconoscimento degli oggetti può essere utilizzato anche per riconoscere strutture più grandi che vanno oltre gli oggetti di dimensioni di tabella; questo procedimento è in particolar modo descritto dal nome Scene Recognition. Pertanto, il nuovo metodo di conversione basato su immagini consente di raggiungere oggetti di dimensioni molto maggiori che possono essere riconosciuti e tracciati con successo.

Plane Tracking

Con il Plane Detection [20], è possibile creare esperienze AR che ancorano accuratamente i contenuti digitali alle superfici con qualsiasi orientamento. Può tracciare piani orizzontali, sopra e sotto (come soffitti o pavimenti), piani verticali o anche piani arbitrati (ad esempio rampe e oggetti in diagonale). Il Plane Detection predispone soltanto della fase di inizializzazione. Durante questa fase

bisogna definire l'altezza del dispositivo da terra. Una volta eseguito tale passaggio, è necessario dare un input in maniera attiva affinché l'algoritmo inizi a tracciare il piano.

QR Code

È possibile creare la funzionalità *QR Code* attraverso l'Image Track [21]. Il *QR code* non viene letto direttamente, bensì tramite l'immagine caricata precedentemente sul database di Wikitude, la quale poi viene riconosciuta e utilizzata nella maniera che si preferisce.

Un'altra possibilità è quella di utilizzare dei plugin, ad esempio ZXing. Si tratta di un'implementazione completa della popolare libreria di codici a barre ZBar nell'SDK di Wikitude. ZBar è una suite di software *open source* per la lettura di codici a barre da varie fonti, come flussi video, file di immagini e sensori di intensità non elaborati.

2.1.4. Valutazione

La ricerca ha portato il sottoscritto a realizzare la [Tabella 2.1](#). Essa mostra le varie tecnologie utilizzate da ogni SDK per arrivare allo scopo finale.

	ARCore	Vuforia	Wikitude
Object recognition	X	Model Target	Simple
			Extended Tracking
Plane tracking	Plane Detection	Ground Plane	Plane Detection
		Mid-Air	
QR Code	Image tracking	Image tracking	Image tracking
		VuMark	

Tabella 2.1: SDK e tecnologie supportate (*Wikitude è stato testato solo su webcam)

Al momento, ARCore, come si è accennato precedentemente, non ha ancora sviluppato la tecnologia per effettuare l'*object recognition*.

Di seguito vi sono altre tabelle realizzate sulla base dell'esperienza accumulata dal sottoscritto testando i vari framework e SDK e utilizzando diversi metodi, oggetti e luci. È stata riportata una valutazione personale testando dei mini-progetti.

La legenda nella [Tabella 2.2](#) è stata utilizzata per descrivere i simboli presenti nelle varie tabelle successive.

Legenda (Quality)	
H	High
M	Medium
L	Low
X	Not recognized

Tabella 2.2: Legenda

Tutti i test riportati successivamente sono stati effettuati con condizioni ambientali ottimali.

Object Recognition

Per l'*object recognition* sono stati scelti tre oggetti con caratteristiche differenti per testare l'efficacia e la precisione della tecnologia al variare della loro semplicità o complessità. Di seguito nella [Tabella 2.3](#) gli oggetti utilizzati sono descritti nel particolare.

Oggetto	Descrizione
Obj Semplice Tinta Unita	Semplice, piccolo, tinta unita
1. Scatola Stroili	
Obj Semplice Multicolor	Semplice, piccolo, rgb
2. Cubo Rubik	
Obj Complesso Multicolor	Complesso, grande, rgb
3. Truck	

Tabella 2.3: Oggetti utilizzati

In questa valutazione non è presente ARCore per il motivo descritto nel [paragrafo 2.1.1](#). I numeri riportati più avanti si riferiscono agli oggetti nella [Tabella 2.3](#).

Luce (1)		
	Vuforia	Wikitude
Artificiale	M/L	X
Naturale	M	L
Poca luce	M/L	X

Tabella 2.4: Confronto condizioni di luce su oggetto 1.

Luce (3)		
	Vuforia	Wikitude
Artificiale	H	L
Naturale	H	L
Poca luce	H/M	L

Tabella 2.5: Confronto condizioni di luce su oggetto 3.

Il primo test riporta due oggetti. La [Tabella 2.4](#) mostra i risultati dei test su un oggetto semplice e monocolore. La [Tabella 2.5](#) mostra i risultati dei test su un oggetto complesso e multicolore. Questi oggetti sono stati valutati in primo luogo con condizioni di luci differenti:

- luce artificiale, l'ambiente è illuminato solo con luce artificiale;
- naturale, l'ambiente è illuminato solo con luce diurna;
- poca luce, l'ambiente è illuminato solo con poca luce naturale.

Si è notato e constatato che, con un oggetto semplice e monocolore, entrambe le tecnologie incontrano difficoltà a riconoscere tale oggetto. Nonostante questa difficoltà, Vuforia riesce a riconoscere l'oggetto anche se non in modo efficiente.

Invece, con un oggetto complesso e multicolore, entrambe le tecnologie migliorano significativamente; in particolare Vuforia, la quale riesce a tracciare l'oggetto in modo efficiente.

Il test esplicita che oggetti più complessi e con più colori risultano alle tecnologie più facili da riconoscere e da tracciare. La condizione di luce non influisce significativamente sul riconoscimento, al contrario della condizione di poca luce, attraverso la quale le tecnologie peggiorano leggermente.

Successivamente, nel secondo test, sono state testate diverse dimensioni.

Dimensioni Multicolor		
	Vuforia	Wikitude
Piccolo (1, 2)	H	L
Grande (3)	H/M	M

Tabella 2.6: Confronto dimensioni differenti

La [Tabella 2.6](#) riporta il test per oggetti di diversa dimensione. Gli oggetti numero uno e numero due sono classificati come piccoli, mentre l'oggetto numero tre come grande. Questo test mostra che Vuforia rimane la migliore come tecnologia, anche se si può notare che con oggetti grandi Vuforia tende a peggiorare leggermente.

Forma Multicolor		
	Vuforia	Wikitude
Semplice (2)	H/M	M
Complessa (3)	H	L

Tabella 2.7: Confronto forme differenti

Come si può notare nella [Tabella 2.7](#), nel terzo test vengono messe a confronto forme diverse. L'oggetto numero due ha una forma semplice (cubo), mentre l'oggetto numero tre ha una forma complessa. Vuforia è sempre migliore rispetto a Wikitude, soprattutto quando si tratta di riconoscere forme complesse. Per quanto riguarda Wikitude, invece, risulta complicato il riconoscimento per via di una non perfetta costruzione dell'oggetto sul cloud.

Come descritto nel paragrafo precedente in riferimento a Wikitude, il suo sistema per riconoscere gli oggetti presenta una prima fase in cui le immagini di ogni lato dell'oggetto vengono caricate sul cloud e una seconda fase in cui il sistema crea un oggetto 3D tramite i dati ricavati dalle immagini. L'ipotesi avanzata è che per Wikitude questa fase sia più complicata e, di conseguenza, alcuni oggetti 3D non possono essere creati correttamente.

Questi due test mostrano che oggetti di grande dimensione e di forma complessa risultano più facili da riconoscere per Vuforia, mentre per Wikitude oggetti semplici e di grande dimensione.

Colori		
	Vuforia	Wikitude
Monocolore (1)	M/L	X
Multicolor (2)	H/M	M

Tabella 2.8: Confronto multicolore e monocolore

Nel quarto e ultimo test, valutato nella [Tabella 2.8](#), viene riportato il confronto di oggetti monocolore e multicolore tra l'oggetto uno, cubo a tinta unita, e l'oggetto due, cubo multicolore. Per entrambe le tecnologie risulta più semplice riconoscere oggetti con più colori.

In conclusione, dopo svariati test, la valutazione personale del sottoscritto è che Vuforia, in quanto *Object Recognition*, è la tecnologia migliore. Il motivo plausibile del fatto che Wikitude non riporta ottimi risultati è che risulta difficile la ricostruzione dell'oggetto 3D dalle foto 2D. Infatti, in alcuni casi, la ricostruzione non avviene in maniera corretta: diverse immagini vengono sovrapposte e alcuni lati dell'oggetto non vengono realizzati. D'altro canto, Vuforia, con la sua applicazione, ottiene un ottimo risultato con la maggior parte degli oggetti.

Plane Track

In questa parte si è confrontata la possibilità di tracciare e riconoscere piani. Prima di iniziare il confronto, è opportuno effettuare delle considerazioni:

- Vuforia (*) non effettua un riconoscimento vero e proprio dei piani, bensì riconosce la superficie tramite Ground Plane ([Figura 2.4](#)), reperibile sul sito ufficiale, e, successivamente, è possibile posizionare un oggetto. È in grado di riconoscere la distanza dal punto di partenza e la scala.
- Per quanto riguarda Wikitude (**), la sua funzionalità non è completamente automatica: ha bisogno di una prima inizializzazione e di settare la distanza da terra del dispositivo. Nonostante questo, riconosce tutte le varie superfici.

Piani			
	ARCore	Vuforia*	Wikitude**
Orizzontale	SI	SI	SI
Verticale	SI	NO	SI
Obliquo	SI	NO	SI

Tabella 2.9: Piani riconosciuti

Il primo test, [Tabella 2.9](#), valuta semplicemente quali tecnologie e quali piani i vari framework riescono a riconoscere. Vuforia in questo caso non è in grado di riconoscere piani verticali e obliqui, al contrario di ARCore e Wikitude.

Luce			
	ARCore	Vuforia*	Wikitude**
Artificiale	H	L	M/L
Naturale	H	L	H/M
Poca luce	H	L	M/L

Tabella 2.10: Confronti riconoscimento piani con condizioni di luce differenti

Il secondo test, mostrato nella [Tabella 2.10](#), verte sulle variazioni di luce descritte precedentemente. Si nota che ARCore ha un buon tracciamento per ogni condizione di luce, mentre Wikitude funziona meglio con la luce naturale. Per quanto riguarda Vuforia, non risulta molto efficace.

In conclusione, in questo ambito Vuforia rimane quasi esclusa, in quanto il suo riconoscimento del piano ha la funzione di essere solo la base per il piazzamento di un qualsiasi oggetto 3D. Di conseguenza, riconosce soltanto un piano alla volta.

La migliore tecnologia in quanto fluidità e precisione è stata realizzata da ARCore, il quale risulta veloce e preciso nel riconoscere diversi piani contemporaneamente. Anche Wikitude esegue il test in maniera soddisfacente; tuttavia, riscontra delle difficoltà nel riconoscere diversi piani con condizioni di luce sfavorevoli.

QR Code

Per il riconoscimento del *QR code* sono state utilizzate delle immagini che possono essere riconosciute da tutti e tre gli SDK. Le immagini scelte sono mostrate nella [Figura 2.8](#) e variano in base al colore, dimensione e complessità.

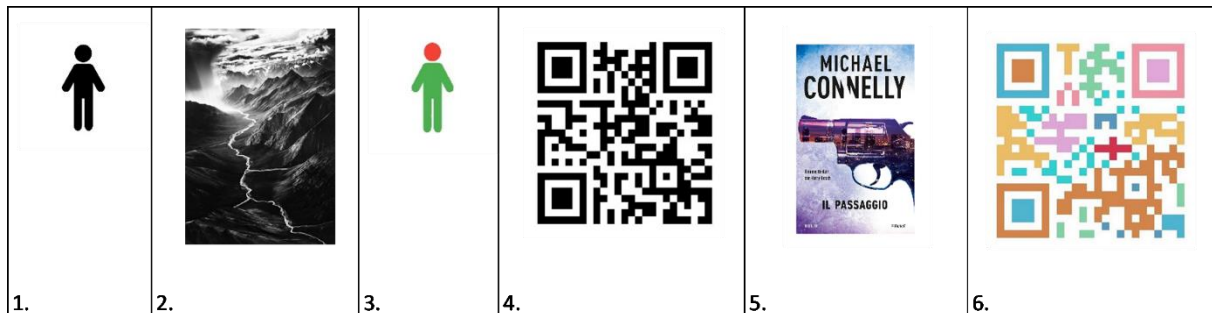


Figura 2.8: Campione di immagini

Valutazione della qualità al variare della luce.

Luce Complesso (2)			
	ARCore	Vuforia	Wikitude
Artificiale	H/M	H	H
Naturale	H	H	H
Poca luce	H	H	L

Tabella 2.11: Confronto condizioni di luce differenti su immagine 2.

Luce Semplice (1)			
	ARCore*	Vuforia	Wikitude
Artificiale	X	H	H
Naturale	X	H	H/M
Poca luce	X	H/M	L

Tabella 2.12: Confronto condizioni di luce differenti su immagine 1.

Nel primo test sono state valutate differenti condizioni di luce, come descritte precedentemente, e con due immagini. Nella [Tabella 2.11](#) viene valutata l'immagine numero due, complessa. Nella [Tabella 2.12](#), invece, viene valutata l'immagine numero uno, semplice.

Si può constatare che tutte e tre le tecnologie hanno un buon tracciamento in quasi tutte le condizioni di luce per l'immagine complessa, mentre risulta più difficile tracciare un'immagine semplice. Questo risultato è dovuto dal fatto che ogni tecnologia individua un numero non sufficiente di punti su un'immagine semplice e lineare.

Tuttavia, rispetto alle altre, Vuforia riesce a tracciare anche l'immagine semplice nelle varie condizioni di luce.

Dimensioni (4)			
	ARCore	Vuforia	Wikitude
Piccolo	L	H	H
Grande	M/L	H	H

Tabella 2.13: Confronto dimensione su immagine 4.

Dimensioni (6)			
	ARCore	Vuforia	Wikitude
Piccolo	H/M	H	H/M
Grande	H/M	H/M	H

Tabella 2.14: Confronto dimensione su immagine 6.

Nel secondo test si è valutata la dimensione su immagini monocolori, [Tabella 2.13](#), e multicolori, [Tabella 2.14](#). Per eseguire questo test sono state stampate un'immagine piccola (3,5cm x 3,5cm) e una grande (11cm x 11cm) sia per il numero quattro che per il numero sei.

Il risultato mostra che sia Vuforia che Wikitude ottengono un ottimo risultato nell'immagine monocolori, mentre ARCore presenta delle difficoltà nel tracciamento. Al contrario, ARCore migliora sul tracciamento dell'immagine multicolore rispetto a Vuforia e Wikitude, le quali peggiorano leggermente.

È possibile notare che la differenza di dimensione non fa variare di molto il riconoscimento.

Gli ultimi due test vertono su immagini semplici monocolori e multicolori e sulla forma semplice o complessa di immagini multicolori.

Colori Semplice			
	ARCore*	Vuforia**	Wikitude**
Tinta unita (1)	X	H	H/M
Multicolor (3)	X	H	H/M

Tabella 2.15: Confronto monocolori e multicolori

Dal terzo test, mostrato nella [Tabella 2.15](#), non risulta alcuna differenza su un'immagine semplice multicolore o monocolori. Vuforia si afferma comunque come miglior tecnologia. Seguono Wikitude, infine, ARCore, che non riconosce nessun'immagine.

Forma Multicolor			
	ARCore	Vuforia	Wikitude
Semplice (3)	X*	H	H/M
Complessa (5)	H/M	H	H

Tabella 2.16: Confronto forma su immagini multicolori

Nel quarto test, [Tabella 2.16](#), si riscontra un netto miglioramento nel riconoscere un'immagine complessa da parte di ARCore e Wikitude, mentre Vuforia svolge un ottimo lavoro in entrambe le situazioni.

Considerazioni:

- (*) L'elaborazione non trova abbastanza punti nell'immagine semplice per essere usata come target.
- (**) Riconosce l'immagine, ma a volte confonde il multicolore con il monocoloro.

Per concludere, nel campo dell'image tracking, le migliori tecnologie appartengono a Wikitude e Vuforia in quanto dispongono di un buon tracking e un buon cloud su cui immagazzinare le varie immagini. Si noti inoltre come Vuforia abbia sviluppato i suoi VuMark, i quali, rispetto all'idea del *QR code*, costituiscono la miglior proposta essendo molto precisi. Complessivamente, per lo scopo del progetto e secondo la personale valutazione del sottoscritto, la tecnologia migliore da poter utilizzare al momento è Vuforia.

2.2. Object Recognition Open Source

L'*object recognition* è una tecnica di visione artificiale per l'identificazione di oggetti in immagini o video. Il riconoscimento di oggetti è un output chiave degli algoritmi di *deep learning* (DL) e *machine learning* (ML). Quando si guarda una fotografia o un video, si riesce immediatamente a individuare persone, oggetti, scene e dettagli visivi. L'obiettivo è quello di insegnare a un computer a svolgere un'attività naturale per l'uomo: identificare e riconoscere il contenuto di un'immagine [22].

È stata effettuata una ricerca più dettagliata per valutare l'*Object Recognition*; la caratteristica di ricerca principale di questo software è che sia *open source*. I framework precedenti, infatti, erano software proprietario. Inoltre, sono stati successivamente valutati rispetto alla precisione, velocità, compatibilità e facilità d'uso. Gli algoritmi Faster R-CNN, YOLO e SSD si sono rivelati come i migliori in questo campo [23]. R-CNN, Fast R-CNN e Mask R-CNN sono stati scartati in quanto Faster R-CNN è un'estensione più veloce dei primi tre. Al fine di testarli e valutarli, di seguito vengono osservati tre software in grado di utilizzare questi algoritmi: Darknet, MediaPipe e TensorFlow.

2.2.1. Darknet



Figura 2.9: Logo Darknet

Darknet [24] è un framework di *neural network open source* scritto in C e CUDA. È veloce, facile da installare e supporta il calcolo di CPU e GPU; inoltre, è ottimizzato per l'*object recognition*. Per effettuare il riconoscimento, si serve di librerie esterne come OpenCV e cuDNN.

OpenCV è una libreria di funzioni di programmazione principalmente finalizzate alla visione artificiale in tempo reale. Originariamente sviluppato da Intel, è stato successivamente supportato da Willow Garage e, in seguito, da Itseez. La libreria è multiplatforma e può essere utilizzata gratuitamente con la licenza *open source* Apache 2.

La libreria NVIDIA CUDA Deep Neural Network (cuDNN) è una libreria di primitive accelerata da GPU per *deep neural networks*. cuDNN fornisce implementazioni altamente ottimizzate per routine standard, come convoluzione in avanti e indietro, *pooling*, normalizzazione e livelli di attivazione.

Darknet viene utilizzato come framework per l'addestramento dell'algoritmo YOLO.



Figura 2.10: Logo algoritmo YOLO

You Only Look Once [6] è una rete che utilizza algoritmi di *Deep Learning* per il rilevamento di oggetti. YOLO esegue il rilevamento degli oggetti classificandoli all'interno dell'immagine e determinando dove si trovano su di essa.

Infine, per rappresentare i risultati, è stata scelta la libreria Barracuda [25], una libreria di inferenza per *neural networks* multiplatforma per Unity.

2.2.2. MediaPipe



Figura 2.11: Logo MediaPipe

MediaPipe [26] offre soluzioni ML personalizzabili multiplatforma per *media live* e *streaming*. Esso offre:

- inferenza ed elaborazione ML veloci, integrate, accelerate anche su hardware comune;
- soluzione unificata funzionante su Android, iOS, desktop / cloud, Web e IoT;
- soluzioni ML all'avanguardia che dimostrano la piena potenza del framework;
- framework e soluzioni entrambi sotto Apache 2.0, completamente estensibili e personalizzabili.

I modelli utilizzabili da MediaPipe sono in formato Tensorflow, in particolare in tensorflow graph (“*.pbtxt”) oppure tensorflow lite (“*.tflite”). L'*object recognition* è stato testato attraverso MediaPipe con un modello tensorflow lite usando l'algoritmo SSD.

Single Shot MultiBox Detector (SSD) è uscito un paio di mesi dopo YOLO, come alternativa. Analogamente a YOLO, il rilevamento degli oggetti viene eseguito in una singola propagazione in avanti della rete. Questo modello CNN *end-to-end* passa l'immagine di input attraverso una serie di livelli convoluzionali, generando caselle di delimitazione candidate da diverse scale lungo il percorso.

La [Figura 2.12](#) mostra l'architettura di Single Shot MultiBox Detector.

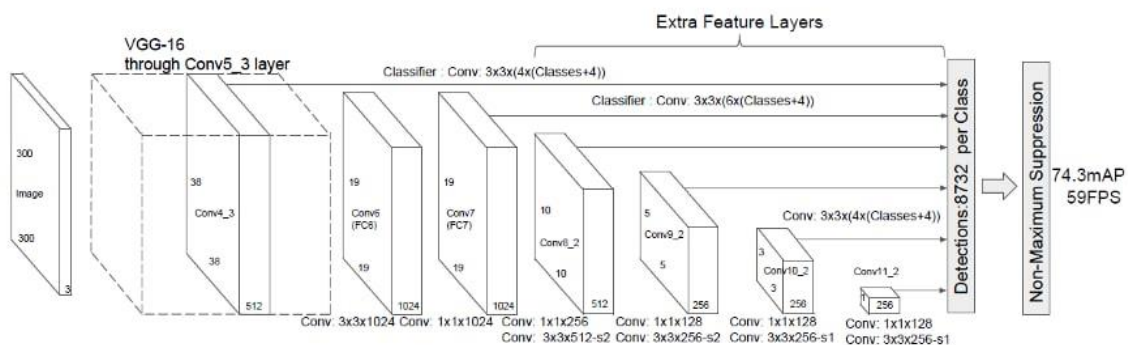


Figura 2.12: Architettura algoritmo SSD (Single Shot MultiBox Detector)

2.2.3. TensorFlow

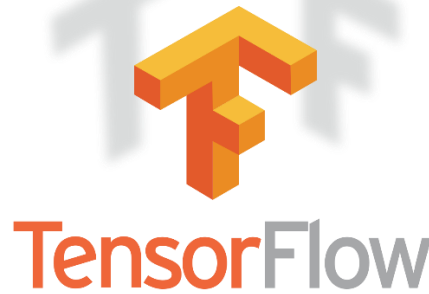


Figura 2.13: Logo TensorFlow

TensorFlow è una piattaforma *open source end-to-end* per il *machine learning*. Dispone di un ecosistema completo e flessibile di strumenti, librerie e risorse della comunità che consente ai ricercatori di spingere lo stato dell'arte nel ML, mentre agli sviluppatori di creare e distribuire facilmente applicazioni basate sul ML.

TensorFlow è stato originariamente sviluppato da ricercatori e ingegneri che lavorano nel team di Google Brain all'interno dell'organizzazione di Machine Intelligence Research di Google per condurre ricerche di *machine learning* e *deep neural network*. Il sistema è abbastanza generale da essere applicabile anche a un'ampia varietà di altri domini. Tensorflow è stato utilizzato per addestrare modelli con Faster R-CNN e SSD. Fast R-CNN risulta essere ancora piuttosto lento, soprattutto perché la CNN è ostacolata dall'algoritmo *region proposal*, la ricerca selettiva. Faster R-CNN risolve questo problema abbandonando il tradizionale metodo di proposta della regione e facendo affidamento su un approccio completamente *deep learning*.

Faster R-CNN divide il quadro di rilevamento in due fasi. Nella prima fase, denominata RPN, le immagini vengono elaborate da un estrattore di caratteristiche e le mappe delle caratteristiche più in alto vengono utilizzate per prevedere le proposte del riquadro di delimitazione. Nella seconda fase, queste proposte vengono utilizzate per ritagliare gli elementi dalle mappe degli elementi più in alto che vengono successivamente inviati alla Fast R-CNN per la classificazione e la regressione del riquadro di delimitazione [27].

La [Figura 2.14](#) mostra l'architettura di Faster R-CNN.

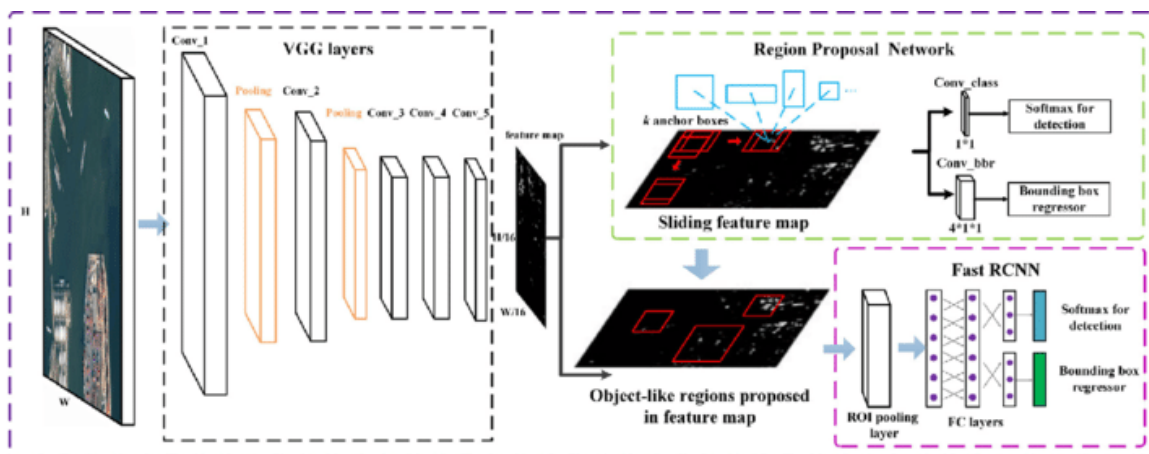


Figura 2.14: Architettura Fast R-CNN

2.3. Scelte per la progettazione

I software descritti precedentemente sono stati analizzati per valutare quale sia la soluzione migliore.

Innanzitutto, MediaPipe, come riportato dal sito ufficiale, risulta in Windows ancora in fase sperimentale; di fatti, sono stati riscontrati alcuni bug nel test. Di conseguenza, è stato testato su una macchina virtuale tramite VirtualBox. I risultati non sono stati soddisfacenti e, dato i molteplici problemi, si è deciso di scartarlo.

Darknet è una soluzione orientata alla precisione e alla velocità. Ad oggi, YOLO è il metodo di rilevamento degli oggetti più popolare per una buona ragione: è in grado di elaborare video in tempo reale con un ritardo minimo, pur mantenendo una precisione rispettabile [23].

Negli algoritmi notiamo invece due categorie principali: gli algoritmi “*Single Shot*” e gli algoritmi “*Two Shots*”.

YOLO e SSD utilizzano un approccio *single shot* in cui la rete è in grado di trovare tutti gli oggetti all’interno di un’immagine in un passaggio (“*One Stage*” o “*Look Once*”). Gli algoritmi a scatto singolo sono più efficienti. Tuttavia, il lato negativo di questi algoritmi è che perdono qualità nella precisione.

Faster R-CNN utilizza un approccio *two shots*, è considerato lo stato dell’arte ed è sicuramente una delle migliori opzioni per il rilevamento di oggetti grazie alla sua precisione. Tuttavia, il rilevamento in *real-time* è ancora lento nelle prestazioni. Il modello di rilevamento *two shots* prevede due fasi: proposta della regione e, successivamente, classificazione di tali regioni e perfezionamento della posizione delle previsioni. Il rilevamento *single shot* salta la fase di proposta della regione, producendo la localizzazione finale e la previsione dei contenuti contemporaneamente.

In conclusione, una caratteristica importante che il modulo richiede è la velocità. Il riconoscimento deve essere veloce perché il modulo ha l’obiettivo di funzionare in *real-time*. Di conseguenza si è dovuto scartare Faster R-CNN per il suo approccio in due fasi. R-CNN si concentra su una regione specifica all’interno dell’immagine e addestra ogni singolo componente separatamente.

Questo processo richiede alla R-CNN di classificare 2000 regioni per immagine, rendendolo molto dispendioso in termini di tempo (47 secondi per singola immagine di prova). Pertanto, non può essere implementato in *real-time*. Inoltre, R-CNN utilizza un algoritmo selettivo fisso, il che significa che non si verifica alcun processo di apprendimento durante questa fase; la rete, quindi, potrebbe generare una proposta di regione inferiore.

YOLO, invece, è molto più veloce (45 fotogrammi al secondo) e più facile da ottimizzare rispetto agli algoritmi precedenti, grazie all’utilizzo di una sola rete neurale per eseguire tutti i componenti dell’attività [28].

Anche SSD utilizza un approccio simile a YOLO; tuttavia, tra YOLO e SSD è stato scelto il primo per lo sviluppo del modulo, utilizzando Darknet. Qui di seguito le ragioni:

- rispetto a SSD, YOLO ha prestazioni migliori nel COCO Dataset;
- Darknet è ottimizzato per il *real-time object detection*.

COCO Dataset, abbreviazione di Common Objects in Context, è un set di dati di riconoscimento/classificazione di immagini di grandi dimensioni.

La [Figura 2.15](#) mostra le prestazioni di differenti algoritmi, in base agli FPS, addestrati sul COCO Dataset, tra cui SSD e Tiny YOLO.

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link

Figura 2.15: Performance algoritmi sul COCO Dataset

3. Dimostratore tecnologico

3.1. Prototipo

Dopo le scelte per la progettazione, è stato creato un prototipo. Per realizzare il prototipo è stato integrato un progetto testato in fase di analisi, il quale è sembrato essere ben strutturato e comprensibile. In seguito, questo progetto è stato adattato allo scopo dell'elaborato, è stato aggiornato e personalizzato. Il progetto originale lo si può visionare in [questa](#) [29] pagina di Github.

Le tecnologie utilizzate nel progetto sono Unity, come piattaforma, su cui sono stati aggiunti come *packages* la libreria Barracuda e la *software suite* WEAVR. Il progetto rielaborato ha la funzione di essere un modulo aggiuntivo per la *software suite* WEAVR. Esso utilizza modelli realizzati con l'algoritmo YOLO addestrato sul framework Darknet.

Le modifiche più importanti sono state:

- l'aggiornamento alla versione 1.1.2 di Barracuda;
- il passaggio da codice sincrono a codice asincrono per migliorare la performance;
- l'aggiunta della funzionalità di lettura del file di configurazione del modello per avere più informazioni sul modello in uso a portata;
- l'aggiunta di alcuni parti di codice e varie personalizzazioni minori.

3.2. Technology Stack

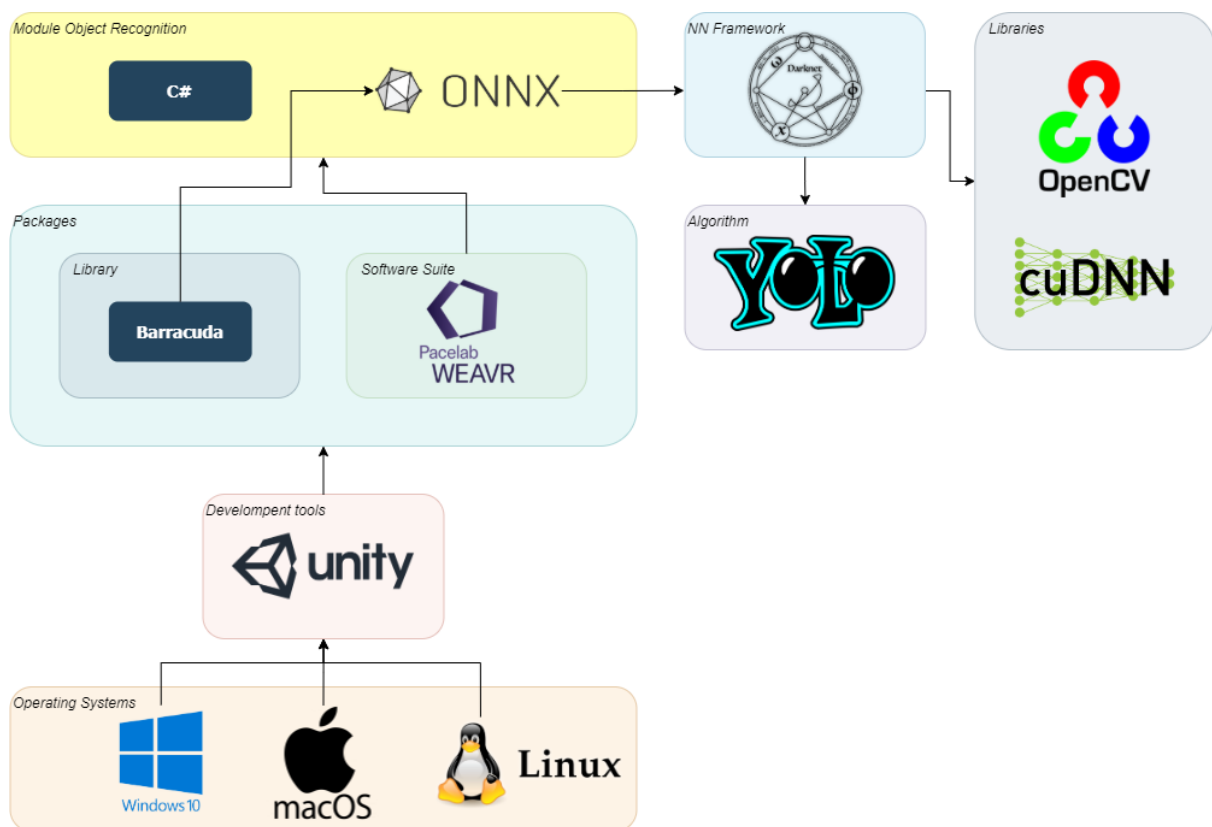


Figura 3.1: Technology stack

La [Figura 3.1](#) mostra le tecnologie utilizzate per la realizzazione del modulo *object recognition*. Il modulo è composto da una parte di codice scritta in C# avente come base il progetto menzionato nel paragrafo 3.1. L'altra parte è composta da uno o più modelli in formato ONNX.

Questi modelli sono realizzati tramite il framework Darknet e l'algoritmo YOLO per l'addestramento. Inoltre, Darknet necessita di altre librerie esterne che sono OpenCV e cuDNN.

OpenCV è una libreria nata e finalizzata alla visione artificiale in *real-time*; mentre cuDNN è una libreria utilizzata per sfruttare al meglio la GPU in occasioni comuni per funzioni comuni nelle *neural networks*.

Il modulo *object recognition*, se approvato, verrà successivamente integrato alla *software suite* WEAVR, utilizzata come pacchetto di Unity insieme alla libreria di inferenza per *neural networks* Barracuda.

La libreria Barracuda è necessaria per la lettura dei modelli creati in formato ONNX.

Quindi, complessivamente, il progetto è stato sviluppato in Unity con due pacchetti principali, ovvero, Barracuda e WEAVR, su una macchina Windows. Tuttavia, è possibile lavorare anche su Linux e Mac OS.

3.3. Modello

Come descritto precedentemente, l'algoritmo scelto per l'addestramento del modello è YOLO.

In particolare, è stata usata la versione 2 di YOLO in quanto è la versione più recente che Barracuda supporta [30].

La [Figura 3.2](#) mostra le architetture supportate da Barracuda.

Supported neural architectures and models

Barracuda currently supports the following neural architectures and models:

- All [ML-Agents](#) models
- [MobileNet v1/v2](#) image classifiers
- [Tiny YOLO v2](#) object detector
- U-Net models
- Fully convolutional models
- Fully dense models
- [Spade](#) architecture

Figura 3.2: Architetture supportate da Barracuda

Precisamente, è stata scelta l'architettura Tiny-YOLO che è circa il 442% più veloce di YOLO, raggiungendo fino a 244 FPS su una singola GPU. Questa scelta si adatta anche alla possibilità di sviluppare un modulo che si possa utilizzare anche su *devices*. Inoltre, Tiny-YOLO ha anche dimensioni ridotte (<50 MB) ed elevata velocità di inferenza che lo rendono naturalmente adatto per i dispositivi mobili.

YOLO è il primo sistema *single shot* di *object detection* (OD). J. Redmon et al. presentano una soluzione innovativa al problema dell'*object detection*, unificando le diverse componenti dei sistemi di OD in un'unica rete. Ciò fa sì che la rete ragioni contemporaneamente sull'intera immagine invece che su delle regioni specifiche, garantendo una maggiore comprensione dell'ambiente in cui le diverse classi di oggetti si trovano. Inoltre, così facendo, crea un legame implicito tra classi di oggetti effettivamente correlate. Come accennato nel capitolo precedente, YOLO si può dividere in tre parti principali: l'algoritmo, il network e la *loss function*.

L'algoritmo

Una volta inserita un'immagine in input, YOLO divide le immagini in una griglia SxS che utilizza per prevedere se la specifica cella contiene l'oggetto (o parti di esso) e, quindi, utilizza queste informazioni per prevedere una classe per essa.

La [Figura 3.3](#) descrive le fasi di YOLO. Il modello, come introdotto precedentemente, divide l'immagine in $S \times S$ celle; per ogni cella predice B *bounding box* (riquadro di delimitazione), ciascuno con un *confidence score* e C classi con le relative probabilità.

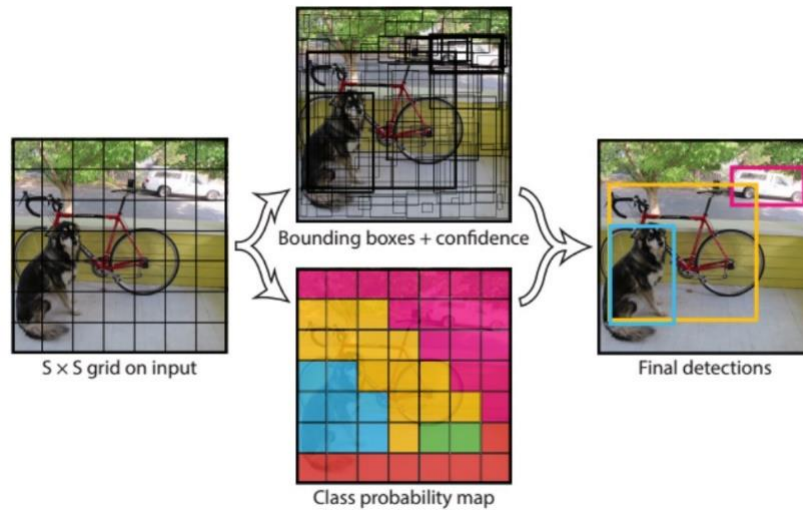


Figura 3.3: Fasi dell'esecuzione di YOLO.

L'algoritmo YOLO utilizza quattro componenti e un valore aggiuntivo per prevedere un output.

- Il centro di un riquadro di delimitazione (b_x, b_y)
- La larghezza (b_w)
- L'altezza (b_h)
- La classe dell'oggetto (c)

Il valore previsto finale è la fiducia (p_c), la quale rappresenta la probabilità dell'esistenza di un oggetto all'interno del riquadro di delimitazione. Le coordinate (x, y) rappresentano il centro del riquadro di delimitazione. In genere, la maggior parte dei riquadri di delimitazione non conterrà un oggetto, quindi è necessario utilizzare la previsione del p_c . Si utilizza un processo chiamato "soppressione non massima" al fine di rimuovere i *bounding box* non necessari con bassa probabilità di contenere oggetti e coloro che condividono grandi aree con altri box.

Il network

Una rete YOLO è strutturata come una normale *convolutional neural network* (CNN): contiene livelli convoluzionali e *max-pooling* e, quindi, due livelli CNN completamente collegati.

La loss function

Soltanto uno dei rettangoli di delimitazione deve essere responsabile dell'oggetto all'interno dell'immagine poiché l'algoritmo YOLO prevede più riquadri di delimitazione per ogni cella della griglia. Per ottenere ciò, si utilizza la *loss function* per calcolare la perdita per ogni vero positivo. Al fine di rendere la *loss function* più efficiente, si seleziona il riquadro di delimitazione con l'intersezione sull'unione (IoU) più alta con la verità di base. Questo metodo migliora le previsioni creando riquadri di delimitazione specializzati che le migliorano per alcune proporzioni e dimensioni.

La [Figura 3.4](#) mostra l'architettura di YOLO. YOLO ha 24 livelli convoluzionali seguiti da 2 livelli interamente connessi. Da notare l'alternanza di livelli convoluzionali di dimensione 1x1, usati per ridurre lo spazio delle attivazioni ottenute dai livelli.

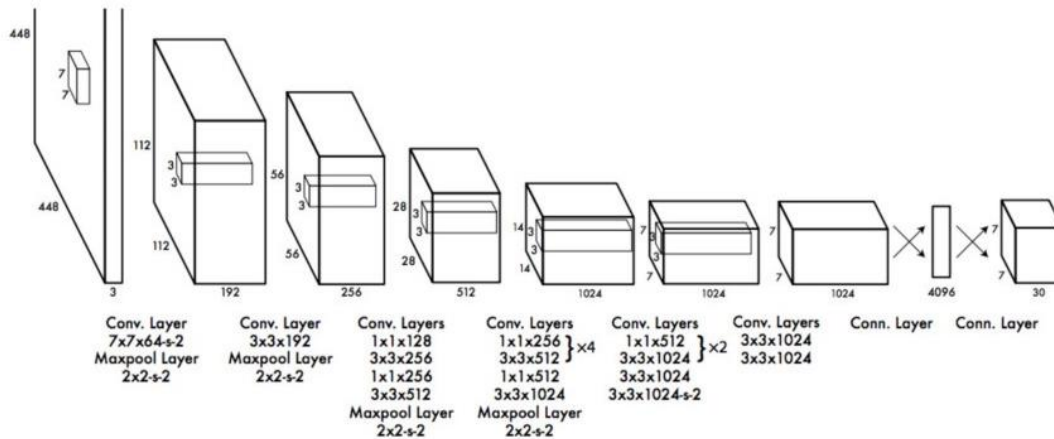


Figura 3.4: Rappresentazione dell'architettura di YOLO.

L'addestramento di YOLO, così come concepito dagli autori, si divide in due fasi: una prima fase di *pre-training*, durante la quale solo i livelli più profondi vengono addestrati, ed una seconda fase di addestramento che coinvolge l'intera rete.

3.3.1. Dataset

Il dataset utilizzato per testare il prototipo è stato creato a partire da 199 foto di 4 carte diverse: asso, cinque, nove e re. Si è cercato di dividere il numero di foto per carte equamente, in modo da trarre il meglio dal modello. Le foto saranno circa 50 per ogni carta, quindi 50 foto per ogni classe.

Per la realizzazione di un modello personalizzato, è necessario dover annotare la posizione del rettangolo di delimitazione nel cui appare una delle classi; per questo scopo è stato utilizzato il software LabelImg [31]. Questo software permette di disegnare manualmente il *bounding box* sull'oggetto e identificarlo tramite la label corrispondente. Le label devono essere state correttamente inizializzate prima di procedere con la fase di annotazione.

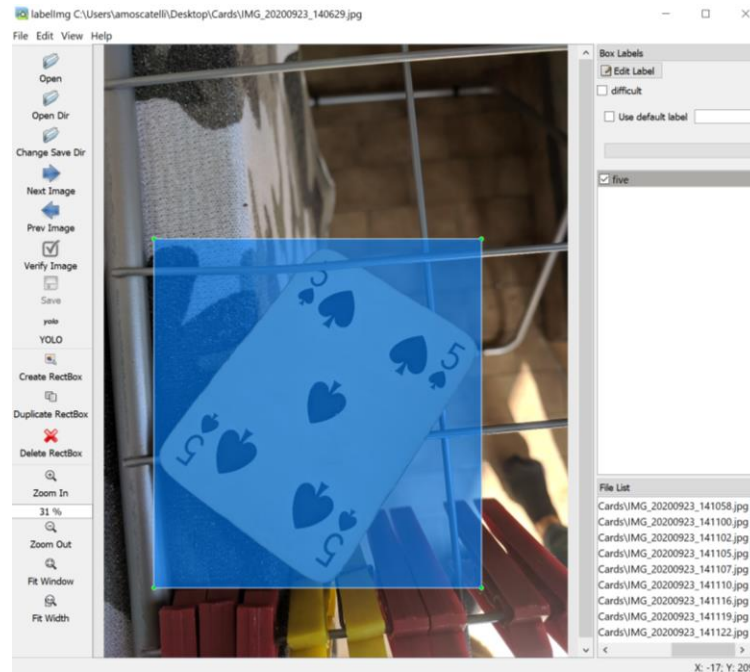


Figura 3.5: Esempio di creazione di un'annotazione con il software LabelImg.

Ci sono due possibilità per salvare le annotazioni: in formato “xml” e formato “txt”. Per YOLO, le annotazioni corrette sono nel formato “txt”.

Il numero minimo consigliato di immagini per avere dei buoni risultati è 200 per classe, dunque, in questo caso si dovrebbe avere a disposizione almeno 800 immagini totali per ottenere un buon risultato. Nonostante questo numero non sia stato rispettato, il modello con il dataset da 200 immagini totali ha ottenuto un discreto risultato; è stato scelto di utilizzare meno immagini al fine di testare le funzionalità del prototipo.

3.3.2. Preparazione training

L'addestramento del modello è stato eseguito completamente sul cloud. Sono stati usati in particolare Google Colaboratory e Google Drive. Questa scelta è stata dettata dal fatto di risolvere il problema di una macchina fisica non adatta all'addestramento di modelli di *machine learning* e *deep learning*.

Se si vuole addestrare il modello sulla propria macchina, è necessario che essa abbia una buona GPU a cui affidare il lavoro parallelo di cui Darknet necessita.



Figura 3.6: Logo Google Colaboratory

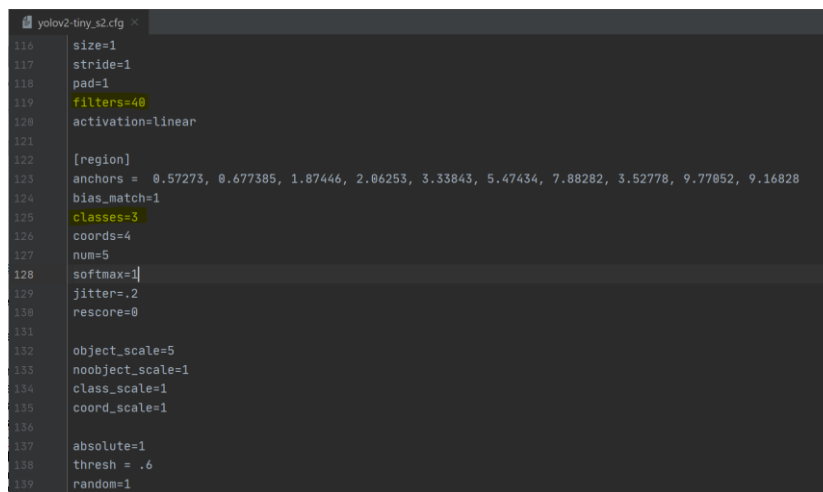
Colaboratory, o in breve “Colab”, è un prodotto di Google Research. Colab consente a chiunque di scrivere ed eseguire codice Python arbitrario tramite il browser ed è particolarmente adatto per l'apprendimento automatico, l'analisi dei dati e l'istruzione. Più tecnicamente, Colab è un servizio per notebook Jupyter in *hosting* che non richiede alcuna configurazione per essere utilizzato, fornendo al contempo accesso gratuito alle risorse di elaborazione, comprese le GPU.

È necessario scaricare sul drive il repository di Darknet per addestrare l'algoritmo. In aggiunta al repository, è necessario caricare la cartella con le foto e annotazioni create durante la preparazione del dataset nel passaggio precedente. Dal sito ufficiale di Darknet è stato scaricato il file di configurazione generale per Tiny YoloV2. È però necessario modificare alcuni parametri qui di seguito descritti:

- alla linea 125, il parametro “classes”, il quale rappresenta quanti oggetti/classi si debbano rilevare; le classi devono coincidere con le label create precedentemente nel momento della creazione delle annotazioni. Nel nostro caso, tale parametro corrisponde a quattro: asso, cinque, nove, re.
- alla linea 119, il parametro “filters” che è un parametro calcolabile tramite “num” e “classes”, deve essere modificato nel seguente modo:

$$filters = (classes + 5) * num .$$

La [Figura 3.7](#) mostra una parte del file di configurazione e i valori da modificare per il *training* dell'algoritmo.



```

116 size=1
117 stride=1
118 pad=1
119 filters=40
120 activation=linear
121
122 [region]
123 anchors = 0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828
124 bias_match=1
125 classes=3
126 coords=4
127 num=5
128 softmax=1
129 jitter=.2
130 rescore=0
131
132 object_scale=5
133 noobject_scale=1
134 class_scale=1
135 coord_scale=1
136
137 absolute=1
138 thresh = .6
139 random=1

```

Figura 3.7: Parte del file di configurazione

Pre-Training

Gli autori di YOLO iniziano l'addestramento allenando solo i primi 20 livelli convoluzionali, a cui aggiungono un livello di *average-pooling* ed uno interamente connesso. Per tale scopo, utilizzano il dataset della competizione ImageNet 1000-class con immagini in input di dimensione 224x224 pixel. Gli autori hanno addestrato il modello per circa una settimana ottenendo un'accuratezza dell'88% circa sul *validation set* di ImageNet 2012.

YOLO-v2 utilizza Darknet-19 *classification network* per l'estrazione dei features dall'immagine di input. L'architettura del classificatore Darknet-19 è schematizzata nella [Figura 3.8](#). Si noti la frequente alternanza di livelli di dimensione 1x1. Lo scopo è quello di ridurre il numero di parametri del modello.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figura 3.8: Architettura della rete Darknet-19.

Inoltre, è necessario compilare Darknet in modo da essere certi di impostare a 1 i valori riferiti alla GPU e a OpenCV nel file “Makefile”, così da utilizzare le librerie necessarie per l’*object recognition*. Successivamente, prima di far partire l’addestramento, è necessario suddividere il dataset in due parti:

- la prima parte, necessaria per la fase di *training*, deve essere la parte più rilevante in cui devono comparire tra il 70 e il 90 per cento di immagini del dataset;
- la seconda parte invece è fondamentale nella fase di validazione dell’algoritmo, essa deve essere tra il 10 e il 30 per cento e complementare alla prima parte.

Quindi, in questo caso, è stato scelto di utilizzare la divisione 75% *training* e 25% *test* per evitare *overfitting* o *underfitting*. Infine, è necessario creare due file, un file “*.data” e uno “*.names”. In breve, nel file “*.data” vengono descritti i percorsi dei file necessari per l’addestramento, mentre nel file “*.names” vengono riportati i nomi di ogni classe o, come precedentemente chiamate, label.

Di seguito la [Figura 3.9](#) mostra i tempi riportati per la creazione del dataset.

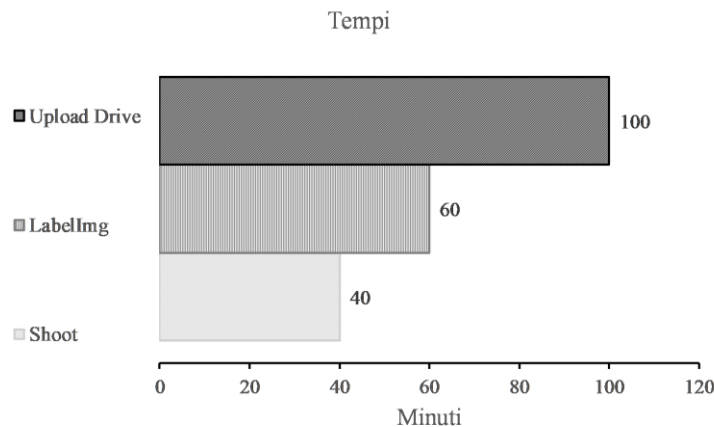


Figura 3.9: Tempi preparazione dataset prototipo

3.3.3. Training

A questo punto, J. Redmon et al. hanno eliminato i due ultimi livelli della rete aggiunti appositamente per la prima parte dell'addestramento per aggiungere altri quattro livelli convoluzionali, seguiti infine da due livelli interamente connessi.

Dato che i modelli per l'*object detection* richiedono informazioni precise per migliorare nell'accuratezza e nella generalità, il modello viene questa volta addestrato con immagini di risoluzione doppia (448x448pixel). Ora è possibile iniziare l'addestramento vero e proprio.

È importante sapere che l'addestramento avviene utilizzando un numero di immagini alla volta. Questo parametro è chiamato *batch*. Ogni batch è diviso in *subdivision*; il numero è possibile settarlo a seconda della portata della macchina su cui avviene l'addestramento. Tuttavia, bisogna mantenere questo numero di *subdivision* in un numero in base due. L'addestramento attraverso Darknet genera dei file di checkpoint, chiamati *weights* o pesi, ogni mille iterazioni.

- *Batch*: Quante immagini e labels vengono utilizzate nel passaggio in avanti per calcolare un gradiente e aggiornare i pesi tramite *backpropagation*.
- *Subdivision*: il lotto è suddiviso in tanti "blocchi". Le immagini di un blocco vengono eseguite in parallelo sulla GPU.

```
3575: 0.116168, 0.139117 avg loss, 0.001000 rate, 0.364080 seconds, 228800 images
Loaded: 13.766554 seconds
Region Avg IOU: 0.800803, Class: 0.995905, Obj: 0.922994, No Obj: 0.007025, Avg Recall: 1.000000, count: 8
Region Avg IOU: 0.800284, Class: 0.979309, Obj: 0.872240, No Obj: 0.007082, Avg Recall: 1.000000, count: 9
Region Avg IOU: 0.804299, Class: 0.975568, Obj: 0.884372, No Obj: 0.007310, Avg Recall: 1.000000, count: 10
Region Avg IOU: 0.805677, Class: 0.968527, Obj: 0.896402, No Obj: 0.007752, Avg Recall: 1.000000, count: 10
Region Avg IOU: 0.797339, Class: 0.997575, Obj: 0.965451, No Obj: 0.008201, Avg Recall: 1.000000, count: 9
Region Avg IOU: 0.793787, Class: 0.912023, Obj: 0.981534, No Obj: 0.008429, Avg Recall: 1.000000, count: 10
Region Avg IOU: 0.811082, Class: 0.937912, Obj: 0.945896, No Obj: 0.008102, Avg Recall: 1.000000, count: 11
Region Avg IOU: 0.812312, Class: 0.997933, Obj: 0.895823, No Obj: 0.008026, Avg Recall: 1.000000, count: 9
```

Figura 3.10: Esempio di batch

La [Figura 3.10](#) è un esempio di un *batch*. In questo caso, ogni batch aggiunge 64 immagini all'addestramento e il numero di *subdivision* è 8. I parametri più importanti sono:

- *Avg loss*: si riferisce all'errore di perdita medio. Esso dovrebbe essere il più basso possibile. Come regola generale, una volta che questo dato raggiunge una media inferiore a 0,06, si può interrompere l'addestramento.
- *Avg Recall*: definito nel codice come "Recall/Count". È una metrica per quanti positivi YOLO ha rilevato rispetto alla quantità totale reale di positivi in questa *subdivision*.
- *Region avg IoU*: media dell'IoU, [Figura 3.11](#), di ogni immagine nella *subdivision* corrente.

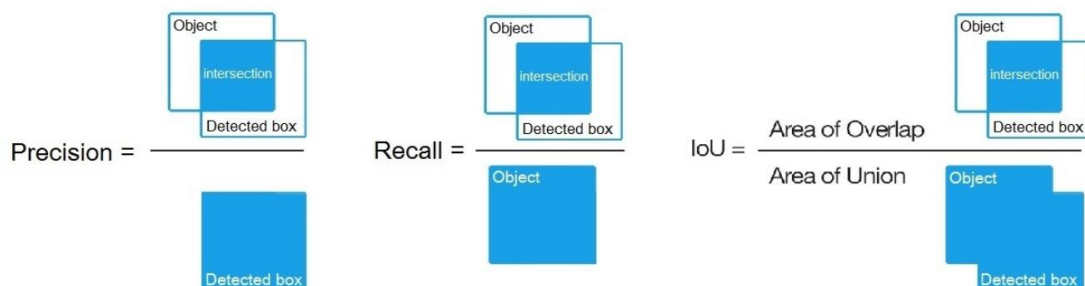


Figura 3.11: Rappresentazione di Precision, Recall e della metrica IOU.

Anche durante questa fase si è tenuto conto dei tempi necessario all'addestramento. Di seguito verrà riportata la [Figura 3.9](#) e, in aggiunta, il tempo necessario al *training* e il tempo totale.

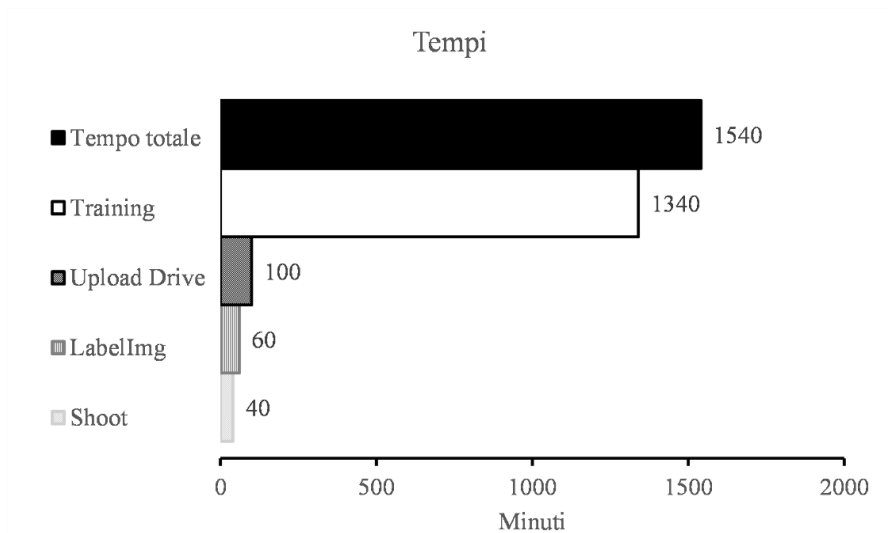


Figura 3.12: Tempi preparazione e training dataset prototipo

Dalla [Figura 3.12](#) si nota che il tempo necessario al *training* è la parte più dispendiosa di questo processo. Infatti, risultano 1340 minuti, circa 22 ore e mezzo, mentre il processo totale dura 1540 minuti, circa 25 ore e mezzo; in questo caso rappresenta l'87% del tempo totale.

3.3.4. Test e valutazione

Lo stop del *training* è possibile dopo esser scesi sotto la soglia dello 0,06 del dato *avg loss* e dopo aver constatato di avere un buon valore nel dato mAP. È possibile anche andare oltre per migliorare il mAP, ma è importante valutare il modello creato. Darknet mette a disposizione anche la funzione “map” la quale calcola e valuta i pesi addestrati. Da questi dati, è possibile condurre una valutazione del modello corrente e decidere se continuare l'addestramento o fermarsi se i risultati sono soddisfacenti.

Di seguito sono stati calcolati i parametri dei vari pesi creati da Darknet ogni mille iterazioni. I dati del migliore risultato ottenuto, che corrisponde a “yolov2-tiny_last.weights”, sono i seguenti:

```
Loading weights from backup_test25/yolov2-tiny_last.weights...
seen 64
Done! Loaded 16 layers from weights-file

calculation mAP (mean average precision)...
52
detections_count = 221, unique_truth_count = 53
class_id = 0, name = ace, ap = 89.79% (TP = 18, FP = 1)
class_id = 1, name = five, ap = 100.00% (TP = 7, FP = 3)
class_id = 2, name = nine, ap = 91.07% (TP = 7, FP = 1)
class_id = 3, name = king, ap = 91.90% (TP = 16, FP = 2)

for conf_thresh = 0.25, precision = 0.87, recall = 0.91, F1-score = 0.89
for conf_thresh = 0.25, TP = 48, FP = 7, FN = 5, average IoU = 71.82 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.931893, or 93.19 %
Total Detection Time: 4.000000 Seconds
```

Figura 3.13: Risultato valutazione di “yolov2-tiny_last.weights”

La [Figura 3.13](#) mostra il punteggio ottenuto dal modello preso in considerazione. Ottiene degli ottimi risultati, soprattutto nel mAP e anche una buona probabilità IoU. Essi informano sul fatto che,

teoricamente, il *bounding box* della previsione dell'algoritmo sarà molto preciso rispetto al *bounding box* di verità.

Di seguito saranno spiegati i tre dati più importanti di questo calcolo che Darknet mette a disposizione.

Il punteggio F1 Score è la media ponderata dei dati *precision* e *recall*. Pertanto, questo punteggio tiene conto sia dei falsi positivi che dei falsi negativi. F1-score risulta preciso anche se si dispone di una distribuzione delle classi non uniforme. Il dato *precision* funziona meglio se falsi positivi e falsi negativi hanno un costo simile. Se il costo dei falsi positivi e dei falsi negativi è molto diverso, è meglio considerare sia il dato *precision* che *recall* [32].

$$F1\ Score = 2 * (Recall * Precision) / (Recall + Precision)$$

La probabilità IoU aiuta a confrontare l'accuratezza delle previsioni dei modelli creati. Usandolo, si può capire quanto bene il riquadro di delimitazione previsto si sovrappone al riquadro di delimitazione della verità di base. Maggiore è l'IoU, migliori sono le prestazioni.

I risultati possono essere interpretati come nella [Figura 3.14](#) sottostante:



Figura 3.14: Rappresentazione IoU

Inoltre, l'IoU, aiuta a rimuovere i riquadri di delimitazione duplicati per lo stesso oggetto. Per questo, si ordinano tutte le previsioni/oggetti in ordine decrescente di confidenza. Se due rettangoli di delimitazione puntano allo stesso oggetto, la loro IoU sarà sicuramente molto alta. In questo caso, si sceglie la casella con maggiore sicurezza e si rifiuta la seconda. Se la probabilità IoU fosse molto bassa, questo potrebbe significare che le due caselle puntino a oggetti diversi della stessa classe (come cani diversi o gatti diversi nella stessa immagine).

YOLO definisce il punteggio di confidenza come probabilità della classe (p_c) * intersezione sull'unione (IoU):

$$Confidence = p_c * IoU$$

Al momento del test, il punteggio di confidenza per riquadro di delimitazione è uno degli output della *neural network*; non viene ricalcolato, ma viene utilizzato per creare l'output finale in base a quali caselle hanno la massima affidabilità.

La probabilità mAP sta per media del dato Average Precision (AP). L'AP fornisce una misura della qualità su tutti i livelli di recall per la classificazione di una singola classe; può essere vista come l'area sotto la curva di precision-recall. Quindi, il mAP è la media degli AP nella classificazione multi-classe. Il mAP è una buona misura della sensibilità della neural network. Di conseguenza, un buon mAP indica un modello stabile e coerente attraverso le soglie di confidenza della differenza.

Infine, per arrivare a modelli ONNX testabili sul modulo, i pesi creati vengono testati attraverso la funzione map di Darknet e quelli con i risultati migliori devono essere convertiti in formato protobuf.

In questo caso è stato utilizzato [Darkflow](#). Successivamente, il modello in protobuf è stato visualizzato in Netron [33], il quale mostra il grafo appartenente al modello. Da qui è possibile leggere gli *inputs* e gli *outputs* necessari per convertirlo in ONNX tramite il repository tensorflow_onnx [34].

Di seguito le Figure [3.15](#), [3.16](#), [3.17](#), [3.18](#) mostrano degli screen durante il riconoscimento in *real-time* del prototipo. Rispettivamente, le prime due figure mostrano delle previsioni corrette, mentre le ultime due mostrano previsioni errate.



Figura 3.15: Esempio con previsioni corrette

Figura 3.16: Esempio con previsioni corrette



Figura 3.17: Esempio con previsioni non corrette

Figura 3.18: Esempio con previsioni non corrette

Il modello in questione è scarsamente popolato per via del dataset da 200 immagini totali per quattro classi da riconoscere. Infatti, questo risultato sicuramente migliorerebbe se si partisse da un dataset con almeno 200 immagini per classe. Si può notare dalle figure precedenti che i *bounding box* creati dalle previsioni sono molto precisi, ricoprono molto bene gli oggetti rilevati, ossia le carte. Lo si poteva prevedere anche dai dati, come si è visto in precedenza. Tuttavia, la precisione nel riconoscere la carta in questione risulta più difficile quando ci sono due carte simili dovuto al fatto dell'addestramento del modello solamente sulle quattro carte scelte e quindi di un dataset scarsamente popolato.

In conclusione, si può affermare che il modello con un dataset ottimamente popolato potrebbe raggiungere un'ottima precisione. Per raggiungere l'ottimo globale si necessita di almeno 200 immagini per classe fino ad arrivare ad un tetto massimo di circa 2000 [24].

Il prototipo con il dataset per riconoscere le quattro carte da gioco non raggiunge risultati previsti dopo il test perché non sempre esegue previsioni corrette in caso ci siano carte simili nell'obiettivo della camera; ad esempio il prototipo con la carta che raffigura il re e la carta che raffigura il jack, messe insieme nell'obiettivo, non effettua previsioni corrette, come si nota dalle Figure 3.17, 3.18. Nonostante questo problema nella realizzazione del modello, il prototipo ha constatato che YOLO è un algoritmo preciso e veloce nel riconoscere gli oggetti e quindi il modello realizzato può essere usato per l'*object recognition*.

There is no minimum images per class for training. Of course the lower number you have, the model will converge slowly and the accuracy will be low.

What important, according to Alexey's (popular forked darknet and the creator of YOLO v4) how to improve object detection is :

For each object which you want to detect - there must be at least 1 similar object in the Training dataset with about the same: shape, side of object, relative size, angle of rotation, tilt, illumination. So desirable that your training dataset include images with objects at different: scales, rotations, lightings, from different sides, on different backgrounds - you should preferably have **2000** different images for each class or more, and you should train **2000*classes iterations** or more

<https://github.com/AlexeyAB/darknet>

So I think you should have minimum 2000 images per class if you want to get the optimum accuracy. But 1000 per class is not bad also. Even with hundreds of images per class you can still get decent (not optimum) result. Just collect as many images as you can.

Figura 3.19: Numero immagini minime consigliate

La Figura 3.19 mostra quante immagini sono necessarie per il dataset. In realtà, si scopre che non esiste un numero minimo o massimo di immagini per classe; tuttavia, sono consigliate circa 2000 immagini per classe, cioè per ogni oggetto che si intende riconoscere in un singolo modello per riuscire ad arrivare all'ottimo globale.

4. Integrazione e valutazione

4.1. Dataset Samsung Galaxy S2

Dato che il WEAVR è una piattaforma per il *professional training*, sulla base del dimostratore tecnologico, si è deciso di creare un modello più professionale che potesse avere e svolgere un compito più verosimile nell'ambito WEAVR. Di conseguenza, sono stati costruiti tre modelli in grado di riconoscere il telefono Samsung Galaxy S2. Questi tre modelli sono stati creati variando il numero di immagini nel dataset per valutare le prestazioni e la precisione ottenute dai modelli creati e per valutare quante immagini sono realmente necessarie per ottenere un buon risultato sul target scelto. Il target ha lo schermo riflettente, una superficie lucida e una dimensione di 7cm x 13cm.

Dalla [Figura 3.19](#) si consigliano circa 2000 immagini per classe per arrivare all'ottimo. Le classi saranno tre: Fronte, Retro e Batteria, per un totale di 6000 immagini. Questo target non è però possibile per via di alcune limitazioni; di conseguenza, si è scelto di arrivare ad un massimo di mille immagini per classe e testare anche la qualità del modello con un minor numero al fine di valutare quante immagini siano realmente necessarie per ottenere dei buoni risultati.

Quindi, si è scelto di addestrare i tre modelli con 50, 200, 1000 immagini per classe, con un totale di immagini rispettivamente di 150, 600, 3000.

Per una possibile replica in futuro, si è realizzata una stima dei costi valutando il tempo per la creazione dei vari dataset e le prestazioni dei modelli ottenuti. Quindi, si è creato il grafico nella [Figura 4.1](#) riguardante i tempi di preparazione del dataset.

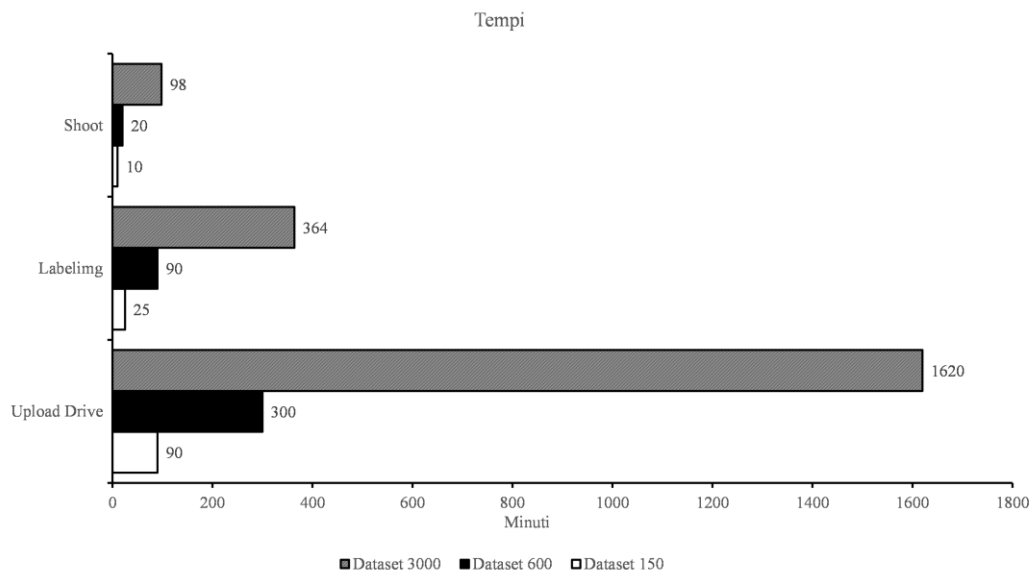


Figura 4.1: Costi preparazione dataset

Questa tabella si basa su un lavoro con limitazioni, come ad esempio una connessione a internet ridotta (circa 10 MB). Inoltre, il caricamento del dataset nel drive può essere eliminato in caso si abbia un pc con buone prestazioni e, secondo i dati raccolti durante il training dell'algoritmo, con almeno 4GB di GPU. In questo caso, si ridurrebbe il costo importante in termini di tempo, così da lavorare direttamente su PC e non sul cloud con Google Colaboratory.

Quindi, come si può constatare, il peso maggiore ricade nell'annotare le immagini attraverso il software LabelImg. Questo costo però è inevitabile: le annotazioni, infatti, sono necessarie all'algoritmo per testare e validare le sue previsioni durante il *training*. Inevitabile è anche il costo del tempo

necessario per scattare le foto, a meno che non si abbia già a disposizione un dataset per un oggetto il quale può essere trovato in qualche dataset online; ad esempio, Google mette a disposizione i suoi [dataset](#) [35]. In questo caso è possibile anche eliminare il tempo per le annotazioni perché esse sono comprese nei dataset. La limitazione corrisponde al fatto che gli oggetti disponibili sono molto generali e, come in questo caso, per oggetti specifici è necessario creare il dataset manualmente.

Di seguito, come mostrato dalla [Tabella 4.1](#), vi è un esempio di come viene scelto il modello nella fase di valutazione da utilizzare nel prototipo. Vengono confrontati tre valori principali descritti nel capitolo precedente: F1-Score, mAP e IoU. Nella legenda sotto la tabella, in verde vengono riportati i risultati sopra alla media, mentre in rosso quelli sotto la media. Come si può notare, vi sono due modelli che stanno con tutti e tre i valori sopra la media, cioè il modello con 16 mila iterazioni e il modello con 20 mila e 500 iterazioni.

Iter (mila)	mAP	F1-Score	IoU
13	84,7	0,82	59,62
14	85,94	0,75	47,95
15	84,63	0,8	56,74
16	88,47	0,82	60,46
17	85,18	0,81	56,74
18	84,65	0,85	66,28
19	84,73	0,8	60,25
20	84,88	0,82	63,92
20,5	88,07	0,84	65,03
21	84,57	0,86	65,37
	Sopra la media		
	Sotto la media		

Tabella 4.1: Esempio valutazione modello

Quindi, confrontando questi due modelli, si possono trarre le seguenti conclusioni: il modello con 20 mila e 500 iterazioni risulta leggermente peggiore nel mAP, ma migliore nel F1-Score e, ulteriormente, nel IoU. Di conseguenza, viene preso quest'ultimo.

Ovviamente, in questa valutazione non è stato tenuto conto del tempo necessario per arrivare all'ennesima iterazione.

Durante la creazione dei dataset, si è riscontrato un problema riguardo a un modello che non si è più migliorato senza scendere sotto alla soglia consigliata; in particolare, per quanto riguarda il modello da 3000 immagini nel dataset. Infatti, per questo dataset, il dato relativo all'*avg loss* non è sceso sotto lo 0,06 consigliato e il massimo ottenuto da questo sono stati i dati seguenti risultati:

- F1-score: 0,78
- mAP: 87,58
- IoU: 54,55

Si può notare un buon mAP e un F1-score discreto, mentre l'IoU è decisamente peggio rispetto agli altri modelli. Quindi, teoricamente, il riconoscimento dovrebbe avvenire, ma il *bounding box* creato non risulterà preciso sull'oggetto da riconoscere.

Il grafico nella [Figura 4.2](#) mostra i punteggi ottenuti dai tre dataset.

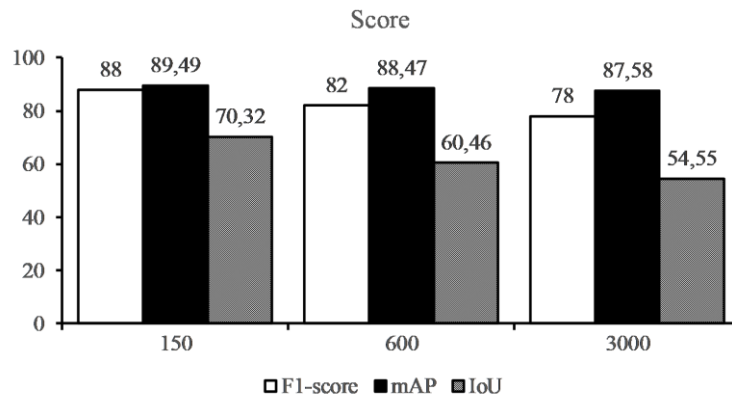


Figura 4.2: Punteggi dataset

Nel *testing* del modello con il dataset da 3000 immagini, il riconoscimento risulta infatti difficile sia per le lunghe e sia per le corte distanze, oltre al fatto che i box di delimitazione risultano, come previsto, imprecisi. Di seguito, è stato realizzato un grafico, [Figura 4.3](#), dando un punteggio in base alla valutazione personale complessiva del sottoscritto riguardo le condizioni diverse di luce e le distanze differenti dei tre diversi modelli e, in aggiunta, è stata eseguita una media di tali punteggi.

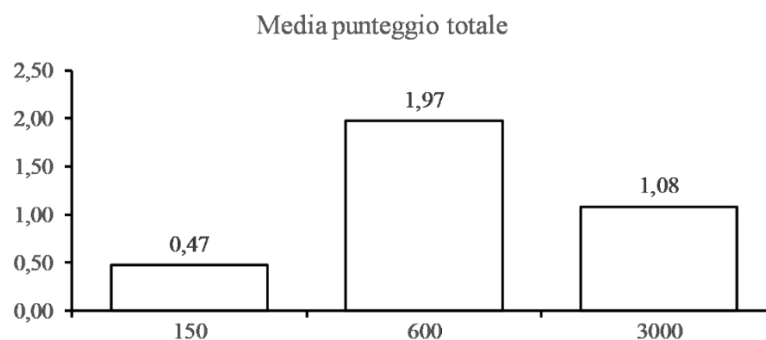


Figura 4.3: Medie punteggi dataset

Questo risultato ha portato ad una verifica delle 3000 immagini e la successiva eliminazione di immagini mosse e sfocate o difficili da riconoscere anche per un essere umano. Quindi, sono state sostituite circa 300 immagini ed è stato addestrato un nuovo modello.

Il nuovo modello ha portato nuovi dati nella [Figura 4.4](#) che, rispetto alla [Figura 4.2](#), sono peggiorati. Nonostante questi peggioramenti, nei test eseguiti, il nuovo modello ha migliorato i risultati, come si potrà vedere successivamente. Anche questo nuovo modello non riesce comunque ad arrivare ad un *avg loss* consigliato sotto lo 0,06; infatti, rimane sulla soglia del 0,1.

Tuttavia, il target scelto rimane un target difficile per via delle varie schermate che riflettono luce e immagini. Lo schermo riflette entrambe, mentre il retro riflette solo la luce. Queste condizioni sono in generale le condizioni più sfavorevoli riguardo l'*object recognition*.

Il grafico nella [Figura 4.4](#) mostra i nuovi dati ottenuti con il nuovo dataset da 3000 immagini.

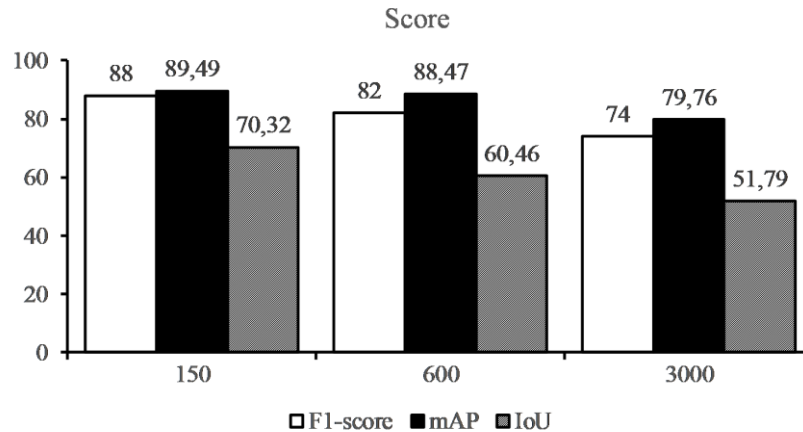


Figura 4.4: Dati ottenuti dai modelli con dataset differenti

Di seguito, la [Figura 4.5](#) riporta i punteggi aggiornati rispetto alla valutazione personale del sottoscritto. Si può notare che il modello da 3000 immagini ha subito un miglioramento nonostante i dati fossero peggiorati. Si ipotizza che sia dovuto al rumore relativo all'inserimento di immagini con diverse varianti di esposizione di luce e di tipologia dell'oggetto. Di conseguenza, le immagini sostituite creavano falsi positivi da cui ne consegue un punteggio più alto nel calcolo di F1-Score, mAP e IoU.

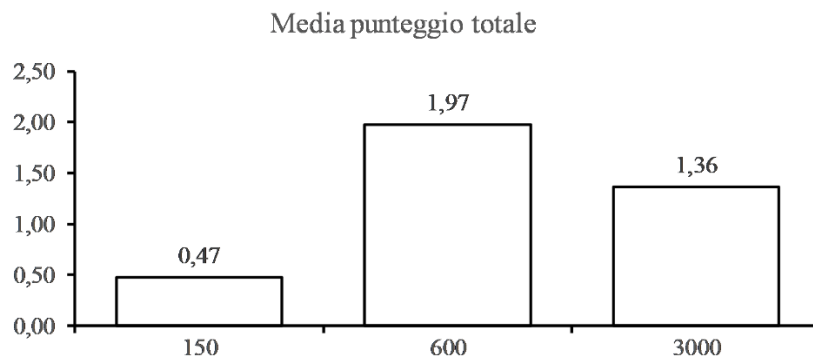


Figura 4.5: Medie punteggi dataset (Dataset 3000 aggiornato)

Questo punteggio, come il precedente, è stato valutato nella maniera seguente. Si è stilata una legenda per rappresentare i risultati attraverso la confidenza che il modello calcola per ogni *bounding box* creato per ogni *frame*. Effettuando questa valutazione in *real-time*, è stato necessario creare degli intervalli per avere una visione più ad alto livello.

I test sul telefono reale attraverso l'applicazione sono stati effettuati variando le condizioni di luce, le distanze dal target e mantenendo uno sfondo bianco dietro al target.

Si è tenuto conto anche dei tempi per valutare il costo. Come si può notare dalla [Figura 4.6](#), il tempo maggiore è occupato dall'addestramento dell'algoritmo a meno che il dataset risulti molto grande, come nel caso del dataset da 3000 immagini. Infatti, quest'ultimo richiede molto tempo, anche nella fase di preparazione del dataset. Si ricorda che gran parte del tempo della preparazione di questo dataset è occupato dal caricamento su Drive, il quale è un costo risolvibile.

Il tempo dedicato al training per i dataset 150, 600 e 3000 è rispettivamente il 93,50%, 84,76% e 72,44%. Questi dati mostrano che più il dataset diventa grande, più aumenta il costo dedicato alla preparazione di esso (se manuale). Si può vedere infatti come per il dataset da 150 il tempo dedicato al *training* è il 93,50% del totale. Mentre il tempo per il dataset da 3000 è il 72,44% del totale.

Dal grafico in [Figura 4.6](#) si può notare in percentuale i tempi necessari per ogni fase della creazione dei vari modelli.

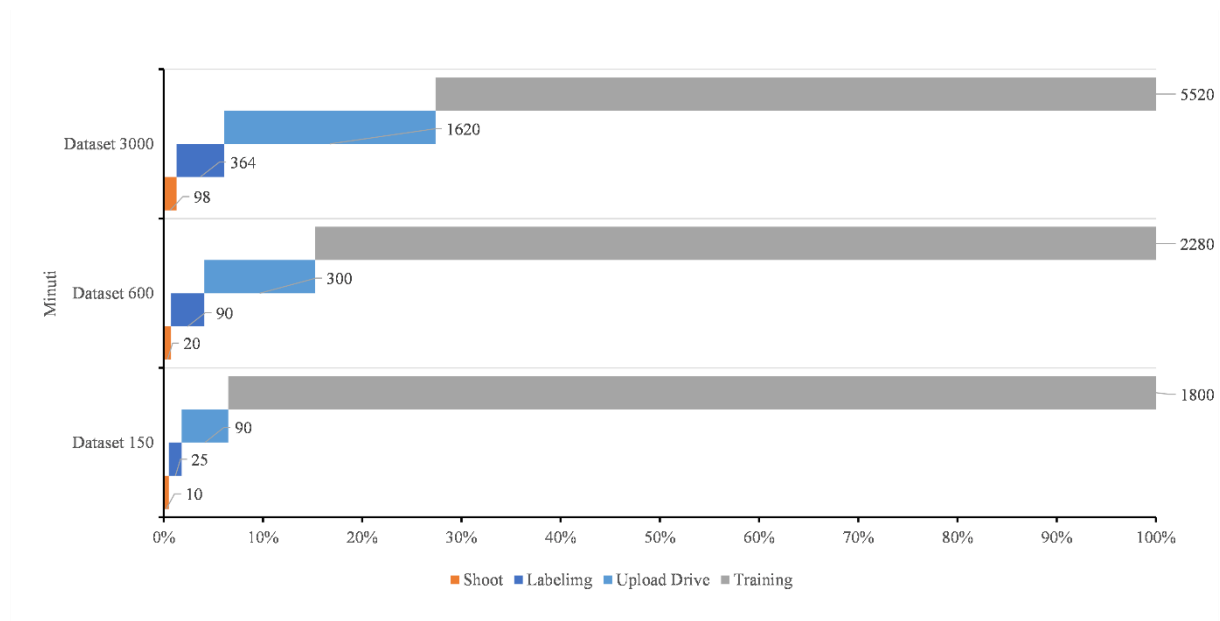


Figura 4.6: Tempo in percentuale per ogni fase

Si può concludere affermando che, il costo maggiore sia quello dedicato al *training* dell'algoritmo, come si può evincere dal grafico nella [Figura 4.7](#) sottostante. Inoltre, si nota anche che più grande è il dataset, più tempo richiede, tanto per la preparazione quanto per l'addestramento.

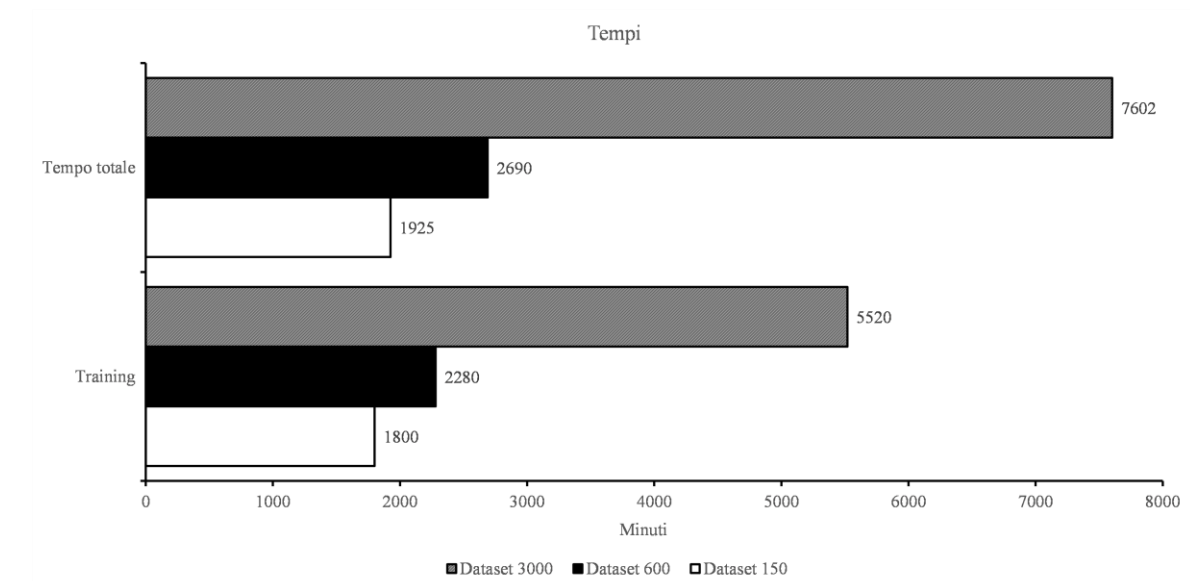


Figura 4.7: Tempo training e totale per la creazione del dataset

Di seguito verranno mostrate alcune immagini dell'*object recognition in real-time*.

Modello con dataset da 150 immagini



Figura 4.8: Riconoscimento fronte (50)



Figura 4.9: Riconoscimento retro (50)



Figura 4.10: Riconoscimento retro aperto (50)

Modello con dataset da 600 immagini



Figura 4.11: Riconoscimento fronte (200)



Figura 4.12: Riconoscimento retro (200)



Figura 4.13: Riconoscimento retro aperto (200)

Modello con dataset da 1000 immagini



Figura 4.14: Riconoscimento fronte (3000)



Figura 4.15: Riconoscimento retro (3000)



Figura 4.16: Riconoscimento retro aperto (3000)

4.2. Valutazione

Per quanto riguarda il modello realizzato tramite dataset da 50 immagini per classe, esso risulta poco preciso e fallisce il test:

- non riconosce sempre il fronte del telefono;
- il retro del telefono viene riconosciuto come fronte;
- il retro aperto del telefono riconosce solo la batteria e non il retro del telefono.

Gli ottimi risultati ottenuti nei dati per il modello da 150 immagini probabilmente riflettono solo una piccola parte delle molteplici possibilità di visione e angolazione del target. Non sono comunque sufficienti per ottenere un modello su più ampia scala e con più vedute. Di conseguenza, i test portano alla conclusione che per questo oggetto 50 immagini per classe non sono sufficienti. Il problema è dato soprattutto dallo schermo che riflette molto la luce e le figure davanti a sé come uno specchio, mentre il retro riflette solo la luce. Questo porta a delle difficoltà all'algoritmo nel determinare quale oggetto e di quale classe fa parte per via di immagini sempre diverse a seconda della luce, posizione e riflesso.

Si prosegue con il modello con un dataset da 600 immagini, cioè da 200 immagini per classe, il minimo di immagini consigliate. Il modello non fallisce completamente il test:

- il fronte del telefono viene sempre riconosciuto;
- il retro chiuso a volte viene confuso con il fronte e a volte lo riconosce, ma non crea perfettamente la dimensione del *bounding box*;
- la batteria viene sempre riconosciuta, ma non il retro del telefono.

Questo test porta alla conclusione che anche 200 immagini per classe sono sufficienti per avere un modello discreto che soddisfi le aspettative, senza però arrivare ad un ottimo risultato.

Il modello con un dataset da 3000 immagini per classe, invece, non dà i risultati sperati. Nonostante l'abbondante numero di immagini, il modello non soddisfa al meglio le aspettative. Ovviamente risulta migliore del modello con un dataset da 150 immagini, ma si rivela peggiore rispetto al modello con il dataset da 600 immagini.

- Il fronte viene sempre rilevato, ma con una confidenza nella media.
- Il retro chiuso viene rilevato con una confidenza medio-bassa e, a volte, viene confuso con il fronte.
- Il retro aperto viene rilevato con una confidenza nella media, mentre la batteria con una confidenza mediamente bassa, a volte assente.

Di seguito verrà mostrato come sono stati testati e valutati i modelli.

Legenda		
Valore	Punteggio	CONFIDENCE
NONE	0	
L	0.25	>40 & <60
M	0.5	>60 & <90
H	0.75	>90

Tabella 4.2: Legenda per punteggi

Si è creata una legenda, [Tabella 4.2](#), per comprendere e poter valutare con dei valori l'efficienza dei modelli.

Modello	Distanza (cm)	Naturale				Artificiale				Poca luce			
		Fronte	Retro	Retro Aperto	Batteria	Fronte	Retro	Retro Aperto	Batteria	Fronte	Retro	Retro Aperto	Batteria
150	20	H	NONE	NONE	M	H	NONE	NONE	M	M	NONE	NONE	L
	40	M	NONE	NONE	NONE	M	NONE	NONE	NONE	NONE	NONE	NONE	NONE
	60	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
600	20	H	NONE	H	H	H	M	H	H	H	L	L	L
	40	H	H	H	L	H	M	M	M	H	M	L	NONE
	60	H	M	M	NONE	H	H	H	NONE	L	L	M	NONE
3000	20	H	M	H	H	M	L	H	M	L	L	M	L
	40	M	M	M	M	L	L	NONE	L	L	L	L	L
	60	L	L	L	NONE	L	L	L	L	L	L	L	NONE

Tabella 4.3: Valutazione alto livello

Dalla [Tabella 4.3](#) si è passati alla [Tabella 4.4](#) con dati numerica per ottenere dei dati valutabili.

LUCE					
Modello	Naturale	Artificiale	Poca luce	Avg luce	Distanza (cm)
150	1,25	1,25	0,75	1,08	20
	0,50	0,50	0,00	0,33	40
	0,00	0,00	0,00	0,00	60
Avg distanza	0,58	0,58	0,25	0,47	
600	2,25	2,75	1,50	2,17	20
	2,50	2,25	1,50	2,08	40
	1,75	2,25	1,00	1,67	60
Avg distanza	2,17	2,42	1,33	1,97	
3000	2,75	2,00	1,25	2,00	20
	2,00	0,75	1,00	1,25	40
	0,75	1,00	0,75	0,83	60
Avg distanza	1,83	1,25	1,00	1,36	
Punteggi sopra la media					
Media tra Avg distanza e Avg luce					

Tabella 4.4: Valutazione numerica alto livello

Si può notare che il modello da 600 immagini ha la maggior parte dei risultati sopra la media. Complessivamente, invece, ha una media di 1,97 su un massimo di 3. Essendo dati estrapolati dalle personali conoscenze del sottoscritto, non possono essere totalmente affidabili, ma la valutazione è stata equa per tutti e tre i modelli.

Dopo i test, infatti, il modello migliore risulta essere il dataset da 600 immagini: esso appare come il più equilibrato nei cambi delle condizioni di luce e di distanza.

4.3. Integrazione WEAVR

Dopo aver testato e valutato il dimostratore tecnologico, anche con il modello in grado di riconoscere il Samsung Galaxy S2, è stata integrata la software suite WEAVR nel progetto di partenza.

È stato modificato il codice al fine di implementare le funzionalità del progetto di partenza e creare una procedura per simulare una situazione possibile.

Nella nomenclatura WEAVR, una procedura può essere una lezione di Virtual Training (VT), un supporto per il supporto operativo (OpS), un Computer Based Training (CBT) o anche un sistema automatico per eseguire la logica. Le procedure vengono create da WEAVR Creator e vengono eseguite da WEAVR Player. La procedura può avere sia un'esecuzione sequenziale che parallela e la sua esecuzione può essere condivisa da più utenti. Inoltre, può anche alterare la sua esecuzione in base all'input dell'utente e / o qualsiasi altro input esterno. Una procedura è progettata per essere autonoma, il che significa che può archiviare tutte le sue risorse con essa, ma può anche ottenere dati esterni durante il *run-time*.

In questo caso, come precedentemente accennato nell'introduzione, è stata creata una procedura OpS in cui vengono spiegati passo per passo i passaggi necessari per svolgerla. In particolare, la procedura riguarda la sostituzione della batteria del telefono Samsung Galaxy S2.

Nella [Figura 4.17](#) viene mostrata la struttura generale di una procedura.

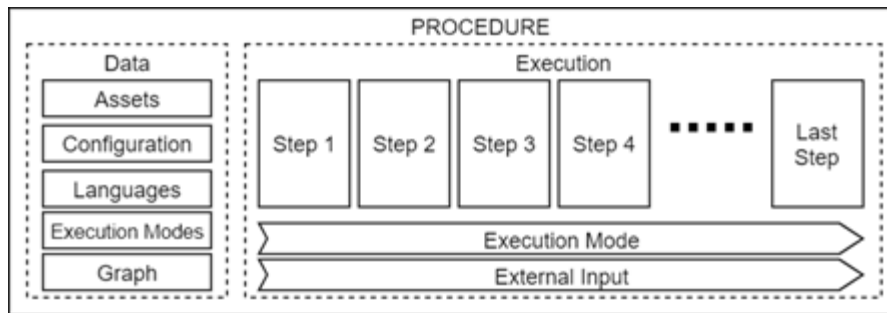


Figura 4.17: Struttura procedura

- *Assets*: gli asset necessari alla procedura per la sua esecuzione (es. File audio, immagini, ecc.).
- *Configuration*: viene utilizzata una configurazione per descrivere come e dove questa procedura può essere eseguita insieme ad alcuni dati aggiuntivi utilizzati dal WEAVR player.
- *Languages*: le lingue supportate da questa procedura; queste sono definite in *edit-time* nel WEAVR Creator e possono essere modificate durante l'esecuzione
- *Execution Modes*: queste modalità definiscono cosa e come deve essere eseguito dalla procedura
- *Graph*: contiene i passaggi e la navigazione tra i passaggi

È stata integrata la scena del Player OpS del WEAVR.

Sono stati aggiunti dei parametri e degli elementi UI per rendere la procedura personalizzata.

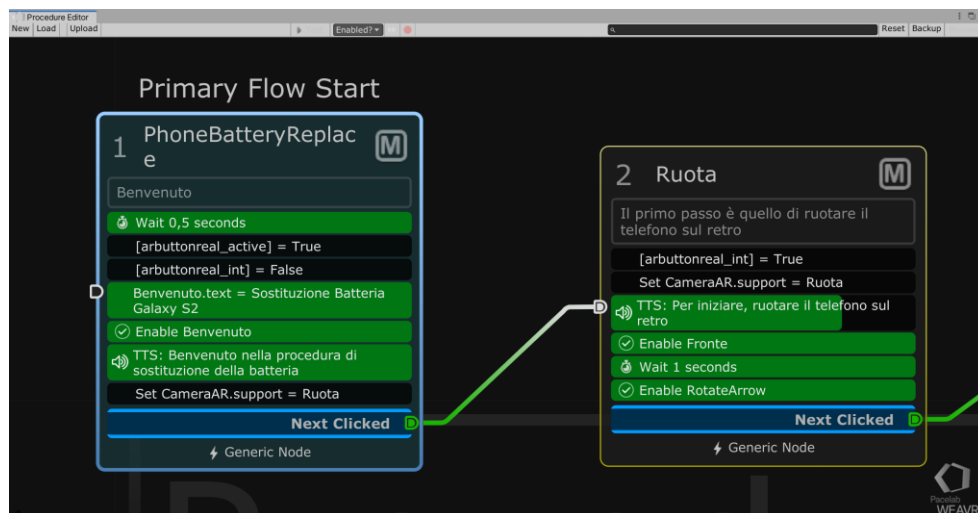


Figura 4.18: Esempio flow della procedura

La [Figura 4.18](#) rappresenta il flusso che segue la procedura OPS del WEAVR. In questa visualizzazione, è possibile modificare e controllare in *real-time* lo svolgimento della procedura e creare flussi separati, come ad esempio flussi primari e flussi secondari. Ogni step può essere raggiunto tramite una condizione imposta o tramite un click di un bottone.

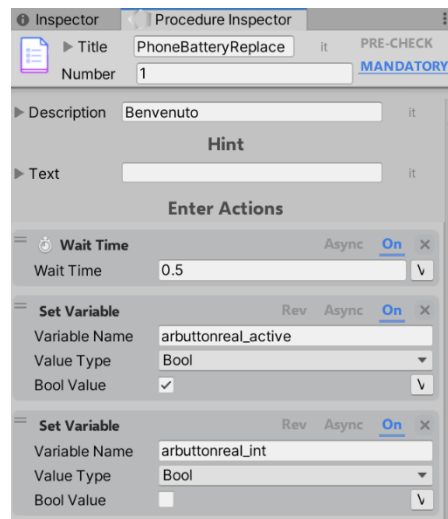


Figura 4.19: Esempio del coding della procedura

La [Figura 4.19](#) riguarda il lato di programmazione relativo alla procedura attraverso il Procedure Inspector. La finestra Procedure Inspector è l'interfaccia che permette di vedere tutti i dettagli degli elementi della procedura (Nodi, Gruppi, Transizioni, ecc.) e di modificarli. La finestra è composta da un'intestazione e da un corpo.

Di seguito, le Figure [4.20](#), [4.21](#), [4.22](#), [4.23](#), [4.24](#), [4.25](#), [4.26](#) mostrano la procedura creata per la sostituzione della batteria del Samsung Galaxy S2 passo per passo.



Figura 4.20: Benvenuto della procedura

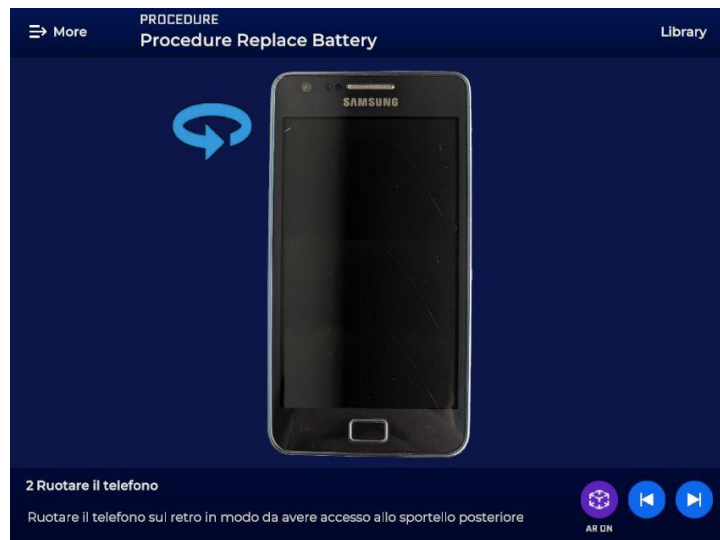


Figura 4.21: Rotazione del telefono

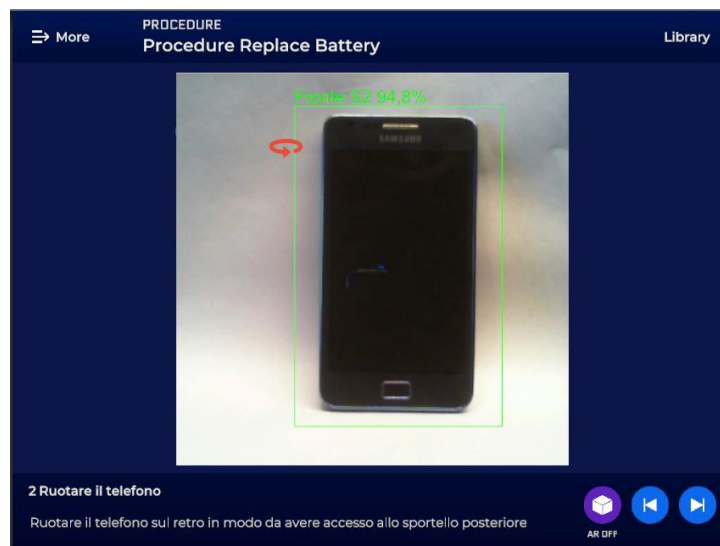


Figura 4.22: Riconoscimento fronte in AR

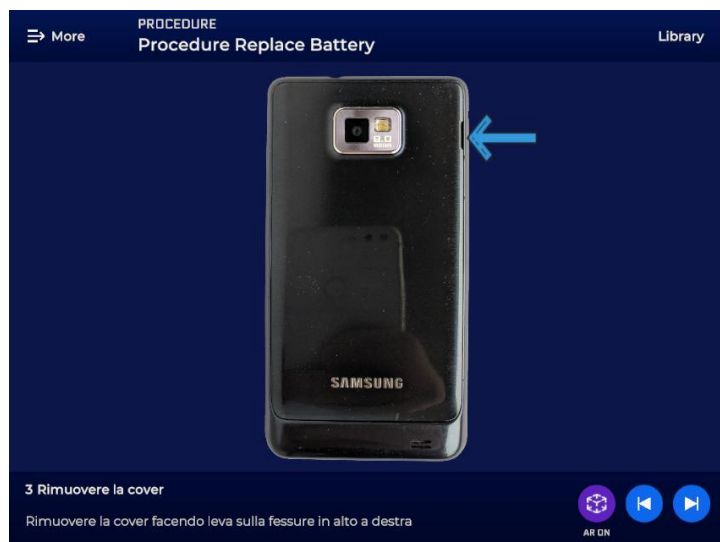


Figura 4.23: Aprire la cover



Figura 4.24: Riconoscimento fessura apertura cover AR



Figura 4.25: Rimuovere la batteria

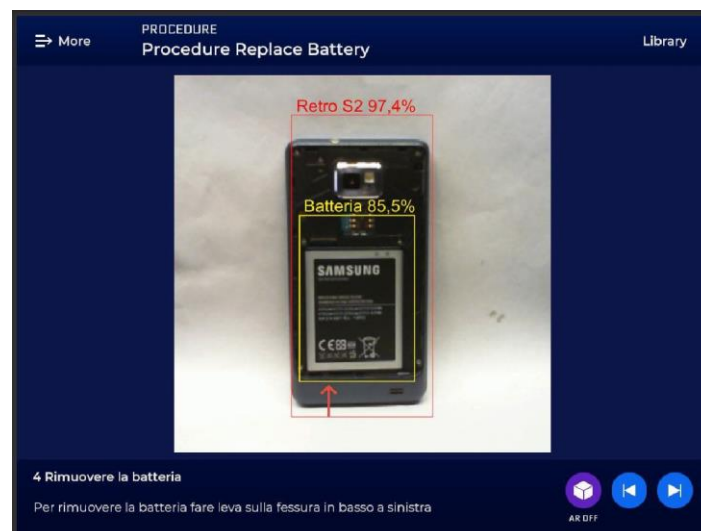


Figura 4.26: Fessura per rimuovere la batteria AR

5. Conclusioni e sviluppi futuri

5.1. Valutazione

Dai test effettuati precedentemente si può arrivare a due conclusioni. La prima riguarda il range di funzionamento ottimale che, per un oggetto di dimensioni circa 7 cm x 13 cm, risulta essere in una distanza dal target che varia tra 20 e 40 cm. È comunque possibile riconoscere il target da una distanza che varia dai 40 ai 60 cm, tuttavia l'algoritmo sarà sempre più impreciso e insicuro sulla confidenza del target riconosciuto. Sopra i 60 cm non è possibile riconoscere questo tipo di oggetto, soprattutto per le dimensioni ridotte. Se il target è di dimensioni ridotte o sempre a grandi distanze, si può migliorare il modello modificando il dato relativo alla dimensione dell'immagine in input nel file di configurazione durante la conversione del modello stesso.

La seconda conclusione riguarda le condizioni migliori di luce, le quali risultano in ambienti con luce naturale oppure anche con luce artificiale, a condizione che il luogo sia ben illuminato. Invece, con poca luce, sia naturale che artificiale, risulta più complicato per l'algoritmo riconoscere l'oggetto. Il dataset necessario e sufficiente per realizzare l'*object recognition* sul target Samsung Galaxy S2 si rivela essere il dataset con 600 immagini complessive.

5.2. Valutazione personale

Innanzitutto, si può notare dai test precedenti che il modello creato appositamente per il prototipo, ovvero addestrato per 7000 steps con un dataset da circa 50 immagini per classe, ottiene un ottimo risultato in termini di F1-score, IoU e mAP rispetto ai modelli realizzati con i dataset rivolti al Samsung Galaxy S2, malgrado uno scarso popolamento del dataset. Questo è causato sia dalle poche immagini nel dataset e, quindi, poche immagini su cui testare il modello, sia da una complessità fisica minore delle carte da gioco in quanto prive di riflessi e non riflettenti rispetto al telefono Samsung Galaxy S2. Quest'ultimo presenta lo schermo che riflette luci e immagini, mentre il retro provoca dei riflessi. Tali caratteristiche mettono in difficoltà l'algoritmo. Queste ultime tipologie di oggetti descritti risultano, infatti, i più complessi da riconoscere per qualunque algoritmo e software.

In conclusione, a partire dall'esperienza condotta, YOLO è un ottimo algoritmo per il rilevamento degli oggetti in quanto risulta preciso anche su superfici non ottimali. L'aspetto negativo è che, oltre al processo di annotazioni necessario per ogni algoritmo se si vuole ottenere un dataset personalizzato, l'addestramento dell'algoritmo richiede molto tempo, ovvero circa 45 ore. Questo dato si riferisce al modello con il dataset da 600. Tuttavia, sono ore in cui è possibile lasciar lavorare il pc senza dover mettere mano manualmente; di conseguenza non risulta un passaggio così negativo. Inoltre, è possibile migliorare questo dato utilizzando una macchina con elevate prestazioni, così da far lavorare il framework Darknet con più risorse in parallelo e, quindi, ridurre il tempo di addestramento. Dunque, risulta essere un buon metodo, veloce e preciso per realizzare l'*object recognition*.

5.3. Sviluppi futuri

Il modulo creato è in revisione da parte del team R&D di TXT; se approvato il codice, verrà ingegnerizzato e, successivamente, integrato al codice del WEAVR in sostituzione delle librerie di Vuforia al momento in funzione in grado di riconoscere gli oggetti. Il modulo verrà ulteriormente esteso in modo da creare delle interfacce di configurazione del modello (upload delle foto) direttamente nel modulo "creator" della piattaforma WEAVR.

Un altro sviluppo futuro potrebbe essere l'utilizzo di una versione più recente di YOLO.

Glossario

- Anchors** Le ancore sono dimensioni iniziali (larghezza, altezza) alcune delle quali (le più vicine alla dimensione dell'oggetto) verranno ridimensionate alla dimensione dell'oggetto - utilizzando alcuni output dalla rete neurale.
- Apache 2.0** La Licenza Apache è una licenza di software libero.
- API** In un programma informatico, con application programming interface (API) si indica un insieme di procedure atte all'espletamento di un dato compito; spesso tale termine designa le librerie software di un linguaggio di programmazione.
- Backpropagation** La retropropagazione dell'errore è un algoritmo per l'allenamento delle reti neurali artificiali, usato in combinazione con un metodo di ottimizzazione come, per esempio, la discesa stocastica del gradiente.
- CAD** Settore dell'informatica volto all'utilizzo di tecnologie software e specificamente della computer grafica per supportare l'attività di disegno tecnico.
- CUDA** CUDA è un'architettura hardware per l'elaborazione parallela creata da NVIDIA.
- cuDNN** La libreria NVIDIA CUDA Deep Neural Network è una libreria di primitive con accelerazione GPU per deep neural networks.
- Deep Learning** Campo di ricerca dell'apprendimento automatico e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso.
- FPS** Fotogrammi per secondo o meglio "frame rate". È la frequenza di cattura o riproduzione dei fotogrammi che compongono un filmato.
- Framework** Architettura logica di supporto sulla quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.
- GPU** L'unità di elaborazione grafica è un circuito elettronico progettato per accelerare la creazione di immagini in un frame buffer, destinato all'output su un dispositivo di visualizzazione.
- ID** Gli identificatori (ID) sono simboli (token lessicali) aventi la funzione di individuare un insieme di dati (o entità).
- IoT** Internet delle cose, nelle telecomunicazioni è un neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi concreti.
- IoU** L'intersezione sull'unione è una metrica di valutazione utilizzata per misurare l'accuratezza di un rilevatore di oggetti su un particolare set di dati.
- Livelli convoluzionali** Livello principale della rete. Il suo obiettivo è quello di individuare schemi, come ad esempio curve, angoli, circonferenze o quadrati raffigurati in un'immagine con elevata precisione.
- Machine learning (ML)** Variante alla programmazione tradizionale nella quale in una macchina si predispone l'abilità di apprendere qualcosa dai dati in maniera autonoma, senza istruzioni esplicite.
- mAP** Il mAP per il rilevamento di oggetti è la media degli AP calcolati per tutte le classi.

- Mission-critical** Un sistema critico è un generico sistema che, in caso di fallimento nel suo funzionamento, può provocare danni non considerati accettabili.
- ONNX** Open Neural Network Exchange è un ecosistema di intelligenza artificiale open source.
- OpenCV** OpenCV è una libreria software multiplatforma nell'ambito della visione artificiale in tempo reale.
- Overfitting** Nell'overfitting ci sono troppi parametri nel modello e un'elevata variabilità della classificazione. Il modello è troppo complesso e sensibile ai dati di training (high variance).
- Repository** Archivio in grado di contenere dati e relativi metadati. Può offrire un sistema di versionamento in grado di tenere traccia delle modifiche effettuate al suo interno.
- Runtime** Runtime o run-time indica il momento in cui un programma per computer viene eseguito, in contrapposizione ad altre fasi del ciclo di vita del software.
- Scena** Le scene contengono gli oggetti del gioco. Possono essere utilizzate per creare un menu principale, livelli individuali e qualsiasi altra cosa.
- SDK** Un software development kit indica genericamente un insieme di strumenti per lo sviluppo e la documentazione di software.
- SLAM** Simultaneous Localization And Mapping.
- UI** L'interfaccia utente è un'interfaccia uomo-macchina, ovvero ciò che si frappona tra una macchina e un utente, consentendone l'interazione reciproca.
- Underfitting** Nell'underfitting ci sono pochi parametri nel modello e un'elevata discrepanza nella classificazione (high bias). Il processo di apprendimento è troppo semplice.
- UWP** La piattaforma universale di Windows è un'architettura/piattaforma applicativa omogenea creata da Microsoft e introdotta per la prima volta in Windows 10.
- VR** Headset per realtà virtuale è un dispositivo che, se indossato, immerge il soggetto nella realtà virtuale.
- XR** Cross Reality, qualsiasi hardware che combina aspetti AR, MR, VR.

Riferimenti bibliografici

- [1] Digi-Capital. *Is Apple, Facebook or Microsoft the future of augmented reality?* URL: <https://www.digi-capital.com/news/2020/02/apple-facebook-microsoft-augmented-reality-forecast/>.
- [2] Statista. *Augmented reality market size worldwide 2017-2025*. URL: <https://www.statista.com/statistics/897587/world-augmented-reality-market-value/>.
- [3] italiaonline. *Realtà Aumentata: funzionamento e applicazioni pratiche*. URL: <https://www.italiaonline.it/risorse/realta-aumentata-funzionamento-e-applicazioni-pratiche-2208>.
- [4] AI4BUSINESS. *Reti neurali: cosa sono e a cosa servono*. URL: https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/#Cosa_sono_le_reti_neurali_artificiali.
- [5] GlobeNewswire. *Neural Network Market to reach \$38.71 billion, Globally, by 2023, Says Allied Market Research*. URL: <https://www.globenewswire.com/news-release/2020/04/02/2010880/0/en/Neural-Network-Market-to-reach-38-71-billion-Globally-by-2023-Says-Allied-Market-Research.html>.
- [6] YOLO. URL: <https://pjreddie.com/darknet/yolo/>.
- [7] 8allocate. *Top 8 SDKs for Augmented Reality Application Development*. URL: <https://8allocate.com/article/top-8-sdks-for-augmented-reality-application-development/>.
- [8] ARCore. *ARCore overview*. URL: <https://developers.google.com/ar/discover>.
- [9] ARCore. *Fundamental concepts*. URL: https://developers.google.com/ar/discover/concepts#environmental_understanding.
- [10] ARCore. *Augmented Images for Android NDK*. URL: <https://developers.google.com/ar/develop/c/augmented-images>.
- [11] Wikipedia. *Vuforia Augmented Reality SDK*. URL: https://en.wikipedia.org/wiki/Vuforia_Augmented_Reality_SDK.
- [12] Vuforia. *Overview*. URL: <https://library.vuforia.com/getting-started/overview.html>.
- [13] Vuforia. *Model Targets*. URL: <https://library.vuforia.com/features/objects/model-targets.html>.
- [14] Vuforia. *Vuforia Object Scanner*. URL: <https://library.vuforia.com/features/objects/object-reco/vuforia-object-scanner-users-guide.html>.

- [15] Vuforia. *Model Target Generator User Guide*. URL: <https://library.vuforia.com/articles/Solution/model-target-generator-user-guide.html>.
- [16] Vuforia. *Ground Plane*. URL: <https://library.vuforia.com/features/environments/ground-plane-guide.html>.
- [17] Vuforia. *Image Targets*. URL: <https://library.vuforia.com/features/images/image-targets.html>.
- [18] Vuforia. *VuMark*. URL: <https://library.vuforia.com/articles/Training/VuMark.html>.
- [19] Wikitude. *Object Tracking*. URL: <https://www.wikitude.com/external/doc/documentation/latest/unity/objecttrackingnative.html#object-tracking>.
- [20] Wikitude. *Plane Detection*. URL: <https://www.wikitude.com/external/doc/documentation/latest/unity/instanttrackingnative.html#plane-detection-experimental>.
- [21] Wikitude. *Image Recognition*. URL: <https://www.wikitude.com/documentation/latest/android/imagerecognition.html#image-recognition>.
- [22] MathWorks. *Object Recognition*. URL: <https://it.mathworks.com/solutions/image-video-processing/object-recognition.html>.
- [23] netcetera. *Object Detection and Tracking in 2020*. URL: <https://blog.netcetera.com/object-detection-and-tracking-in-2020-f10fb6ff9af3>.
- [24] Darknet. URL: <https://pjreddie.com/darknet/>.
- [25] Barracuda. URL: <https://github.com/Unity-Technologies/barracuda-release>.
- [26] Mediapipe. URL: <https://google.github.io/mediapipe/>.
- [27] ResearchGate. *Multi-scale object detection in remote sensing imagery with convolutional neural networks*. URL: https://www.researchgate.net/figure/The-architecture-of-Faster-R-CNN_fig2_324903264.
- [28] missinglink. *YOLO Deep Learning: Don't Think Twice*. URL: <https://missinglink.ai/guides/computer-vision/yolo-deep-learning-dont-think-twice/>.
- [29] wojciechp6. *YOLO-UnityBarracuda*. URL: <https://github.com/wojciechp6/YOLO-UnityBarracuda>.
- [30] Barracuda. *Supported neural architectures and models*. URL: <https://docs.unity3d.com/Packages/com.unity.barracuda@1.1/manual/SupportedArchitectures.html>.
- [31] LabelImg. URL: <https://github.com/tzutalin/labelImg>.

- [32] Exsilio. *Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures*. URL: <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/>.
- [33] Netron. URL: <https://netron.app/>.
- [34] Tensorflow-onnx. URL: <https://github.com/onnx/tensorflow-onnx>.
- [35] Google. *Open Images*. URL: <https://storage.googleapis.com/openimages/web/index.html>.