Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Scuola di : Scienze informatica Corso di Laurema magitrale in: informatica per il management

Framing Federated Learning: Sviluppo di un Framework per l'Analisi delle Architetture Centralizzate e Distribuite

Relatore:
Prof. ANGELO TROTTA

Presentata da:
GIACOMO DI VAIRA

Sessione estiva
2023-2024

Alla mia Mamma

Indice

Introduzione

Nel corso degli ultimi anni il numero di applicazioni basate su meccanismi di intelligenza artificiale (AI) ha avuto un notevole incremento. Questo ci mette dinnanzi ad un futuro in cui potranno essere sviluppati una nuova generazione di prodotti e servizi, in cui i sistemi informatici saranno sempre più "autonomi" ed in grado di adattarsi alle esigenze di ogni utilizzatore. La prospettiva che ci si attende dal crescente sviluppo del campo informatico basato sull'intelligenza artificiale è quindi quella di avere a disposizione, nei prossimi anni, delle applicazioni in grado di aiutarci nella risoluzione di un gran numero di problemi. Questa crescita è fortemente sostenuta dal diffondersi di strumenti come sensori, accessori intelligenti ed altri dispositivi che basano il loro funzionamento sulla raccolta di una gran mole di dati, da cui estrapolano "conoscenza" tramite algoritmi basati sull'AI.

Al contempo, però, bisogna tener conto anche dell'esistenza di alcune problematiche alle quali possono dover far fronte sia gli utilizzatori che gli sviluppatori.
Sebbene chiunque si dedichi alla creazione di applicazioni possa creare un'applicazione che sfrutta un motore basato sull'intelligenza artificiale, al contempo è da tenere in considerazione il fatto che i modelli di AI spesso richiedono una componente hardware molto prestante. A gravare su questa situazione contribuiscono sia gli attuali problemi riscontrati nell'industria di fabbricazione di micro-conduttori che il continuo incremento della domanda di dispositivi quali le GPU (Graphics Processing Unit), rendendo queste componenti molto costose e difficili da reperire sul mercato.
Per quanto concerne gli utilizzatori, invece, il problema principale riguarda la tutela della loro privacy spesso violata da questi sistemi di AI. Per funzionare coerentemente e quindi per adattarsi al meglio al proprio utilizzatore, gli algoritmi di AI hanno infatti bisogno di una gran numero di dati, come suddetto, che possono essere sensibili o non di publico dominio. Da questa necessità di raccolta informazioni scaturisce l'obbligo da parte degli utilizzatori di tali tecnologie di condividere i propri dati, i quali vengono raccolti in grandi dataset utilizzati all'occorrenza per l'addestramento dei modelli. Ovviamente la centralizzazione dei dati implica numerosi rischi di divulgazione ed utilizzo improprio. Di fatto queste informazioni vengono spesso usate, oltre che per l'addestramento dei modelli da cui l'utente si aspetta di trarre principalmente benefici, anche per altri scopi, tra cui quello di addestrare modelli che permettono, ad esempio, di identificare strategie di marketing più efficienti attraverso la segmentazione dei clienti.

Questo lavoro di tesi nasce quindi dal desiderio di approfondire le problematiche appena enunciate, oltre che di analizzare possibili metodologie per superarle. Nello specifico verrà trattata una nuova e recente tecnica applicata nel campo dell'AI che prende il nome di Federated Learnig (FL), per mezzo della quale si tenta di ovviare a tali limiti (privacy,

calcolo computazionale). Nella mia tesi ho deciso di studiare questo nuovo paradigma di apprendimento, il cui funzionamento verrà presentato nel capitolo 1, focalizzando l'attenzione sui passaggi che questo prevede e discutendo i molteplici vantaggi e limiti che ne conseguono.
Il Federated Learning (FL) ha come obbiettivo quello di superare i limiti imposti dall'addestramento con database centralizzato ("classico"), infatti propone di addestrare modelli di AI evitando la condivisione dei dati, al fine di garantire la privacy degli utilizzatori del modello. Oltretutto questa tecnica di addestramento rientra tra quelle di training distribuito, sebbene necessiti di fissare alcuni vincoli iniziali.
Nel contesto del Federated Learning l'addestramento viene eseguito dai diversi sistemi (worker/utenti che decidono di partecipare al FL) che, tramite i propri processori, eseguono in un primo momento un training sui soli dati a loro disposizione. Al termine di questo primo step, i worker, condividono con il server centrale esclusivamente i risultati dell'addestramento.
La struttura classica di FL che la letteratura propone maggiormente per questo approccio prevede, come appena spiegato, che vi sia un server al centro dei diversi worker (centralized federate d learning), il quale si occupa di raccoglie i dati ottenuti dall'addestrmento del modello sui diversi partecipanti e di gestire la coordinazione tra questi. Per di più, in un secondo momento, il server applica operazioni di aggregazione su questi dati al fine di ricondividerli nuovamente con gli stessi worker, che possono infine procedere nel ripetere nuovamente l'addestramento.
Oltre a questo modello di FL centralizzato, in letteratura si parla spesso di una seconda e differente architettura. Quest'ultima prevederebbe l'esclusione del server, lasciando quindi ad ogni singolo worker l'onere di occuparsi sia dell'addestramento che della creazione del modello globale da condivide, riservandogli al contempo una gran serie di vantaggi. Ovviamente in questa tipologia di FL (decentralized federated learning) è necessario aggiungere un'ulteriore logica per permettere al sistema federato di funzionare correttamente.
Nonostante questa tipologia di addestramento sia un tema molto recente, in letteratura sono già reperibili differenti framework che permettono di implementare un sistema di questo tipo.
Nel presente elaborato ho deciso di proporre una nuova tipologia di framework che chiamerò "SmartFed", grazie alla quale mi è stato possibile analizzare al meglio ogni possibile criticità che si può presentare nell'implementazione di un addestramento federato.
Nel capitolo 4 ho voluto illustrare in che modo è possibile implementare un sistema federato, sia centralizzato che non, sfruttando "SmartFed". L'obiettivo è stato poi quello di verificare l'efficienza dell'addestramento federato e la sua resistenza a possibili cam-

biamenti nella rete di partecipanti. A tal fine, per chiarificare e validare la mia tesi, nel quinto ed ultimo capitolo ho deciso di illustrare ed analizzare i risultati salienti dei test eseguiti.

Per realizzare gli esperimenti, è stato necessario l'utilizzo di un modello di AI, ed io, nelle mie simulazioni ho deciso di servirmi di YOLO (You Only Look Once).
Yolo è un popolare algoritmo di rilevamento oggetti noto per la sua spiccata velocità e precisione. Ho prediletto questo algoritmo poichè molteplici studi certificano le sue prestazioni, inoltre la community che si è creata intorno ad esso mi ha permesso di comprendere al meglio il suo funzionamento. Troverete maggiori specifiche a riguardo nel capitolo 2, dove ne ho illustrato varie caratteristiche, oltre che provenienza, architettura e funzionamento.
Ultimo, ma non per importanza, aspetto che ho dovuto affrontare prima di procedere con gli esperimenti è stato quello della selezione del dataset. Era infatti necessario adoperare un dataset che imitasse il più fedelmente possibile un reale sistema federato, che offrisse quindi uno scenario realistico. Dopo alcune ricerche ho deciso di utilizzare il dataset Visdrone, del quale ho trattato nel capitolo 3. Questa scelta è stata dettata principalmente da due ragioni:

- **L'obiettivo del dataset**: il dataset Vis-drone rappresenta una sfida affascinante per i suoi realizzatori, in quanto richiede l'identificazione di vari tipi di oggetti all'interno delle immagini. Questo compito si presta perfettamente all'utilizzo di YOLO, un potente algoritmo di rilevamento degli oggetti che si basa sulla localizzazione e la classificazione simultanea degli stessi. L'obiettivo è quindi quello di addestrare YOLO utilizzando il dataset Vis-drone, consentendo all'algoritmo di apprendere e riconoscere efficacemente gli oggetti di interesse.

- **Gli strumenti utilizzati per la raccolta dei dati**: il dataset Vis-drone è stato creato attraverso la raccolta di immagini provenienti da una varietà di dispositivi, inclusi droni, elicotteri e aerei. Gli strumenti utilizzati per la sua creazione includono quindi diverse piattaforme di acquisizione di immagini aeree. Questa diversità di fonti di dati ha contribuito a fornire una vasta gamma di prospettive e contesti nelle immagini raccolte. I creatori del dataset hanno infatti affermato che l'obiettivo principale era quello di raccogliere dati realistici e rappresentativi del mondo reale, in modo da fornire un set di dati robusto per lo sviluppo e la valutazione di algoritmi di visione artificiale. La combinazione di diverse modalità di acquisizione delle immagini ha consentito di coprire una vasta gamma di scenari e di oggetti di interesse.

Il mio obiettivo era quello di testare l'efficacia del federated learning utilizzando un dataset originariamente suddiviso su diversi nodi. Ho creato una simulazione in cui ho suddiviso il dataset Vis-drone in diversi host virtuali, che rappresentano una virtualizzazione dei dispositivi che hanno effettivamente raccolto i dati. Successivamente, ho coinvolto questi host virtuali in un sistema federato e ho avviato il processo di addestramento di YOLO su ciascuno di essi.
Questo approccio mi ha permesso di trarre conclusioni sul funzionamento e l'efficacia di questo nuovo paradigma di apprendimento. La suddivisione del dataset in diversi host virtuali e l'addestramento distribuito mi ha permesso di valutare come il federated learning può influire sulle prestazioni e sui risultati dell'apprendimento automatico.
Attraverso queste simulazioni, sono stato in grado di esplorare le dinamiche del federated learning, di valutare le sue capacità e di testare il corretto funzionamento di "SmartFed".

Capitolo 1

Federated Learning

1.1 FL visone d'insieme

Il Federated Learning è un paradigma di apprendimento automatico che consente di addestrare modelli di machine learning . Ciò che distingue questa tecnica di apprendimento automatico dalle altre è che i modelli vengono addestrati su dati che sono distribuiti su dispositivi mobili o localizzati in diverse posizioni geografiche, senza la necessità quindi di trasferire i dati raccolti da ogni dispositivo ad un server centrale. Questo fa si che gli utilizzatori possano costruire un buon modello di AI senza uno scambio di dati privati, superando quindi problemi critici quali la sicurezza e la privacy di dati personali e sensibili. Il federated learning è dunque opposto alle tecniche di apprendimento automatico centralizzate, le quali prevedono che i dati vengano caricati su un server, o ai più tradizionali metodi decentralizzati secondo cui i dati locali sono distribuiti in modo identico ed omogeneo.
Con l'uso del federated learning il modello viene addestrato in modo uniforme, distribuito sui dispositivi degli utenti, utilizzando tecniche di collaborazione federata. Si tratta dunque di un paradigma innovativo che mira a superare alcune delle sfide attualmente presenti nel campo dell'AI, rendendo l'apprendimento dei modelli, basato su reti neurali, più "semplice" e accessibile. Una delle caratteristiche fondamentali di questa metodologia è la capacità di distribuire il carico computazionale sui differenti partecipanti all'addestramento federato, consentendo a chiunque di creare e addestrare un modello anche con una complessità molto elevata su differenti dispositivi.

Un processo di Federated Learning (FL) segue la seguente struttura:

1. Preparazione dei dati: i dati vengono raccolti dai dispositivi degli utenti partecipanti al FL. Questi dati vengono memorizzati in locale solo sui dispositivi dei partecipanti all'addestramento federato;

2. Selezione del modello: viene scelto un modello di machine learning appropriato per il task specifico che si intende affrontare. Il modello individuato servirà come punto di partenza per l'addestramento collaborativo durante il processo di FL;

3. Distribuzione del modello: il modello iniziale viene distribuito ai dispositivi locali degli utenti partecipanti, in modo che l'addestramento possa avvenire in modo omogeneo sui dispositivi coinvolti;

4. Addestramento locale: i dispositivi addestrano il modello in locale utilizzando i dati disponibili su ciascuno di questi. L'addestramento avviene, come specificato, in modo locale, ovvero senza condividere i dati con un server centralizzato;

5. Invio dei parametri: dopo aver addestrato il modello locale, i worker inviano al server centrale solo i parametri del modello aggiornati, anziché i dati originali (centralized federated learning).

6. Aggregazione dei modelli: i parametri aggiornati vengono aggregati per creare un nuovo modello globale. Questa aggregazione può avvenire in diversi modi, ad esempio calcolando la media dei pesi dei modelli locali o utilizzando altri algoritmi di aggregazione(cfr cap 1.4).

7. Feedback e iterazioni: il modello globale viene restituito ai dispositivi locali per eseguire ulteriori iterazioni. Questo processo di addestramento e aggregazione viene ripetuto per un numero desiderato di iterazioni o fino al raggiungimento di un determinato criterio di convergenza;

8. Valutazione e validazione: il modello globale viene valutato e validato utilizzando dati di test o di validazione per vagliare le sue prestazioni. Questo aiuta a valutare l'efficacia del processo di FL e ad identificare eventuali miglioramenti o aggiustamenti necessari;

9. Implementazione del modello: una volta ottenuto un modello globale addestrato, può essere implementato per essere utilizzato su nuovi dati o in un ambiente di produzione, secondo le esigenze specifiche del problema da risolvere.

Nella seguente immagine viene mostrato come potrebbe apparire un processo di federated learning centralizzato:

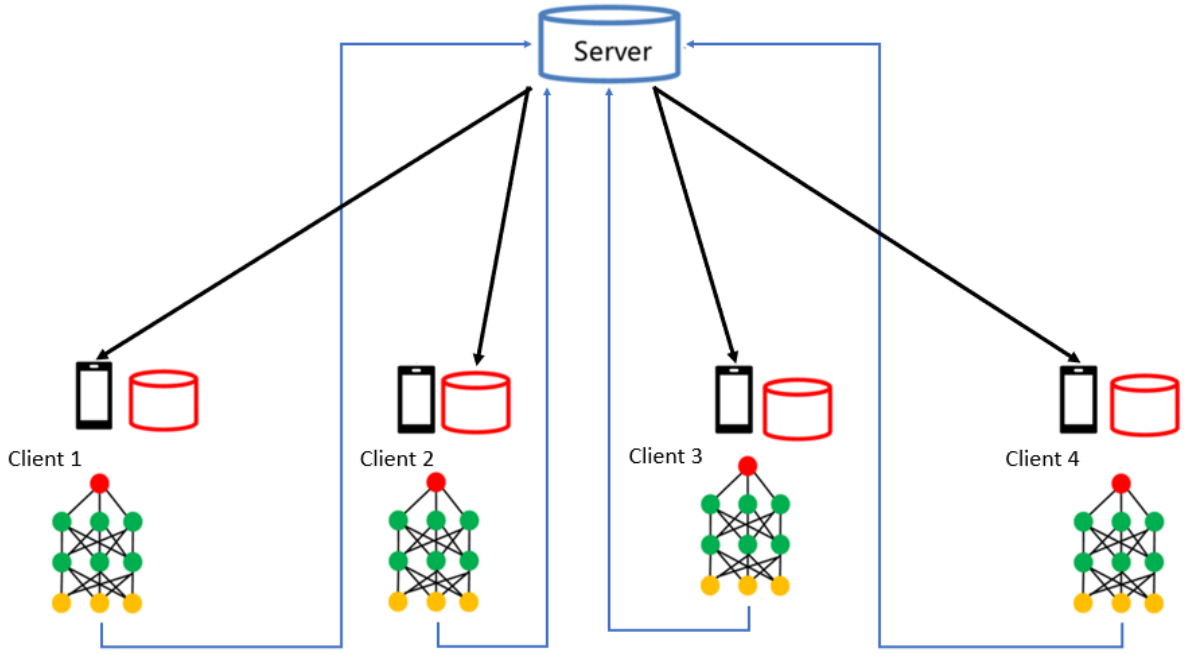

Figura 1.1: Evoluzione del federated learning.

Il federated learning potrebbe essere incluso nella più grande macroclasse delle tecniche di apprendimento basate sul Distributed learning, ossia, una metodologia di addestramento che tenta di dividere il carico computazionale su differenti nodi in modo da rendere meno gravoso il consumo di risorse su un singolo host. Analizzando più nel dettaglio le caratteristiche del FL possiamo identificare due aspetti fondamentali che lo differenziano dalle altre metodologie di distributed learning:

1. In primis non prevede la condivisione di nessun dato raccolto dai worker, in altri modelli di addestramento distribuito non vengono presentate restrizioni su come i dati debbano essere gestiti. Inoltre, come già spiegato in precedenza, anche il FL verticale talvolta richiede la condivisione di alcune informazioni grezze tra le entità coinvolte. Di fatto il punto di forza del FL orizzontale, ancor più che per le altre tipologie, è proprio questo, ossia la possibilità di garantire la privacy per i proprietari dei dati, che per nessun motivo dovranno mai scambiarli con altri dispositivi.

2. In secondo luogo, ma non per importanza, sfrutta le risorse derivanti dall'elaborazione distribuita in dispositivi che potenzialmente sono situati in regioni geografiche e/o organizzazioni differenti. Quando si fa riferimento all'addestramento distribuito

in generale, ci si riferisce alla possibilità di utilizzare più risorse per l'addestramento di un modello. Di solito però queste risorse appartengono allo stesso server (es. differenti GPU) o ad un cluster di nodi. Il FL vuole decentralizzare totalmente il processo di addestramento, permettendo a chiunque ed in qualsiasi area geografica di addestrare il modello con le proprie risorse hardware, qualunque esse siano.

Oggi, l'addestramento di modelli attraverso l'utilizzo del FL viene utilizzato in numerosi campi ed aree proprio per i vantaggi che offre sia a livello computazionale che per la sua caratteristica principale di mirare alla preservazione dei dati dei partecipanti all'addestramento. Basti pensare ad esempio a tutti quegli scenari in cui vengono generati dei modelli di AI addestrati su dati fortemente confidenziali e dei quali si deve quindi garantire la non divulgazione.
Alcuni esempi in cui il FL oggi vede la sua applicazione possono essere:

- Analisi dei dati sanitari distribuiti (2): l'apprendimento federato orizzontale può consentire la collaborazione tra diversi ospedali o enti sanitari per addestrare modelli di apprendimento automatico su dati sensibili senza la necessità di condividere direttamente i dati grezzi.

- Classificazione delle immagini distribuita: anche in questo caso l'apprendimento federato può essere utilizzato per addestrare modelli di classificazione delle immagini su dispositivi mobili o sensori distribuiti, consentendo la privacy delle informazioni e riducendo la necessità di trasferire grandi quantità di dati.

- Previsione del traffico: l'apprendimento federato può essere anche applicato per addestrare modelli di previsione del traffico stradale utilizzando dati provenienti da sensori distribuiti o applicazioni di navigazione, consentendo così una migliore comprensione del flusso del traffico senza dover condividere informazioni specifiche sugli utenti o la loro posizione.(12)

- Suggerimenti di ricerca: può contribuire a migliorare la qualità dei suggerimenti di ricerca da tastiera virtuale senza accesso diretto ai dati utente sottostanti. Anche in questo utilizzo permette comunque di preservare la privacy delle conversazioni degli utenti.(11)

1.1.1 Modalità di FL

Esistono due possibili "modalità" di FL: orizzontale e verticale.
L'apprendimento federato orizzontale, presenta tutte le caratteristiche introdotte nel capiotlo (cap 1.1), ed inoltre le diverse entità possiedono gli stessi insiemi di feature, ma con dati differenti. Questo tipo di apprendimento federato (orizzontale) affronta le sfide della condivisione di informazioni tra entità, al fine di costruire un modello globale che abbia una buona capacità predittiva su tutti i partecipanti, pur rispettando la privacy e mantenendo i dati locali in ciascuna entità. Come spiegato in precedenza può dunque trovare ampia applicazioni in scenari in cui i dati sono distribuiti tra diverse entità e la privacy dei dati è una prerogativa.

Nell'apprendimento federato verticale, al contrario di quello orizzontale, i dati distribuiti tra i diversi partecipanti all'addestramento presentano feature diverse ma correlate. In questa configurazione, infatti, le entità partecipanti hanno insiemi di dati che si sovrappongono solo parzialmente, tuttavia combinandoli è possibile ottenere una visione più completa e arricchita dei dati complessivi. L'obiettivo dell'apprendimento federato verticale è consentire alle entità di addestrare un modello congiunto senza dover condividere direttamente i dati sorgente. Questo è particolarmente importante quando si lavora con dati sensibili o proprietari che richiedono protezione e riservatezza. Piuttosto che inviare i dati stessi, le entità partecipanti condividono informazioni statistiche o aggregazioni che possono essere rilevanti per l'addestramento del modello, come ad esempio le correlazioni tra le feature o le distribuzioni dei dati. L'apprendimento federato verticale trova applicazioni in vari settori, come la sanità, il marketing, la finanza e altri contesti in cui diverse entità devono collaborare per trarre vantaggio dai dati con feature complementari senza dover condividere i dati completi. Questo approccio consente di massimizzare l'utilità e l'informazione dei dati senza compromettere la riservatezza e la proprietà dei dati stessi.

Per chiarire meglio la distinzione tra le due modalità di apprendimento federato possiamo individuare tre punti cardine fondamentali:

- **Configurazione dei dati:**
  - **Apprendimento federato orizzontale:** in questo approccio diversi dispositivi, o entità, possiedono dati locali che coprono diverse istanze (righe) dello stesso insieme di variabili (colonne). Ad esempio, potrebbe trattarsi di diversi ospedali (entità) che possiedono dati sanitari dei pazienti (righe), ma ogni ospedale ha informazioni(colonne) su un sottoinsieme di pazienti.
  - **Apprendimento federato verticale:** qui, diversi dispositivi, o entità, possiedono dati locali che coprono le stesse istanze, ma ciascuno possiede diverse

variabili. Ad esempio, un dispositivo potrebbe avere informazioni sulla cronologia di acquisto di un cliente, mentre un altro dispositivo potrebbe avere informazioni sulle preferenze di prodotto di quel cliente.

- **Collaborazione tra entità:**
  - **Apprendimento federato orizzontale:** gli enti partecipanti condividono solo i parametri del modello addestrato, sui loro dati locali, con un server centrale che coordina l'aggregazione dei parametri. Non vengono quindi condivisi direttamente i dati grezzi tra i partecipanti.
  - **Apprendimento federato verticale:** gli enti partecipanti collaborano per combinare i dati verticalmente, unendo variabili diverse possedute da diverse entità. Di solito, questa collaborazione richiede la condivisione di alcune informazioni grezze tra le entità coinvolte.
- **Scenari di utilizzo:**
  - **Apprendimento federato orizzontale:** è adatto per scenari in cui diversi dispositivi o entità hanno dati simili o correlati, ma la distribuzione dei dati è orizzontale. Ad esempio, può essere utilizzato per addestrare modelli di apprendimento automatico su dati sanitari provenienti da diversi ospedali.
  - **Apprendimento federato verticale:** è idoneo per scenari in cui diversi dispositivi o entità possiedono diverse informazioni su un singolo insieme di istanze. Ad esempio, può essere utilizzato per combinare dati per la personalizzazione di raccomandazioni o analisi di dati multi-fonte.

In questo lavoro di tesi utilizzeremo ed analizzeremo un caso d'uso di FL orizzontale, in quanto i miei esperimenti si sono incentrati sulla valutazione dell'efficacia di questo tipo di addestramento, rispetto ad un addestramento "classico"(dataset centralizzato).

1.2 Definizione formale e Architetture

Nel paragrafo precedente abbiamo presentato una possibile architettura di un sistema di federated learning nell'immagine 1.1.
In questa sezione, invece, tenteremo di scendere nel dettaglio e di formalizzare quello che è il processo standard di federated learning, facendo una distinzioni su due possibili architetture.

Definizione di FL: Siano N proprietari di dati $F_1, \ldots, F_N$, i quali desiderano addestrare un modello di apprendimento automatico consolidando i rispettivi dati $D_1, \ldots, D_N$. Un metodo convenzionale consiste nel mettere insieme tutti i dati e utilizzare $D = D_1 \cup \ldots \cup D_N$ per addestrare un modello MDC(modello con dataset centralizzato). Un sistema di apprendimento federato è un processo di apprendimento in cui i proprietari dei dati collaborano per addestrare un modello $MFDD$(modello federato dataset decentralizzato), nel quale ogni proprietario di dati F_i non espone i propri dati D_i agli altri. Quindi non avviene la creazione del dataset D, dove $D = D_1 \cup \ldots \cup D_N$. Inoltre, l'accuratezza di $MFDD$, indicata come Acc_{MFDD}, dovrebbe essere molto vicina alle prestazioni di MDC che possiamo esprimre con Acc_{MDC}. Formalmente quindi, sia δ un numero reale non negativo, dovrà verificarsi che $|Acc_{\text{MFDD}} - Acc_{\text{MDC}}| < \delta$, dove con delta indichiamo la differenza in termini di prestazioni(accuratezza) dei due modelli .

Come intuibile nella definizione presentata, in cui non viene menzionata nessuna entità al centro dei vari partecipanti all'addestramento federato, è possibile non avere un'entità centrale che si occupi dell'aggregazione dei modelli. Possiamo quindi distinguere due possibili architetture di federated leatning, FL centralizzato (CFL) e FL decentralizzato (DFL).

1.2.1 Apprendimento federato centralizzato

Il FL centralizzato (CFL) 5.4 è una delle architetture di FL più popolari. Come mostrato nella Figura 1.1 , un sistema CFL(centralized federated learning) per eseguire l'addestramento del modello necessita di un server centrale e di un insieme di client. In una singola iterazione di training, tutti i client eseguono l'addestramento del modello fornitogli dal server utilizzando i propri dataset. Successivamente, tutti i client trasmettono i parametri addestrati al server centrale che li aggrega utilizzando un algoritmo di aggregazione. Quindi, una volta calcolato il modello globale viene nuovamente inviato a tutti i client per la successiva iterazione di addestramento. Alla fine del processo di addestramento, ogni client ottiene un modello globale identico oltre al proprio modello personalizzato. Nel CFL, il server è considerato il componente chiave della rete, per coordinare l'aggregazione e la distribuzione degli aggiornamenti del modello ai client.
In questo modo vengono garantite la sicurezza e la privacy dei dati di addestramento in quanto le uniche informazioni condivise con il server sono il risultato degli addestramenti eseguiti da ogni singolo partecipante.

I vantaggi e gli svantaggi dell'utilizzo del CFL sono :

Vantagi:

- **Coordinazione centralizzata:** Il server centrale può gestire in modo efficiente la comunicazione, l'aggregazione dei dati e la sincronizzazione del processo di addestramento.
- **Facilità di implementazione:** CFL può essere più facile da implementare poiché richiede meno complessità nell'organizzazione della comunicazione tra i nodi periferici.
- **Controllo del modello globale:** Il server centrale ha un controllo diretto sul modello globale e può prendere decisioni sulla sua evoluzione.

Svantaggi:

- **Latenza e larghezza di banda:** CFL richiede una comunicazione frequente tra i nodi periferici e il server centrale, il che può comportare problemi di latenza e utilizzo di una banda molto larga.
- **Dipendenza dal server centrale:** La centralizzazione può creare un singolo punto di fallimento e dipendenza da un server centrale affidabile. Che in caso non dovesse essere disponibile potrebbe imputare il fallimento dell'intero processo di addestramento.

1.2.2 Apprendimento federato decentralizzato

FL decentralizzato (DFL)5.4 a differenza del CFL, il DFL è una topologia di rete senza alcun server centrale a coordinare il processo di addestramento. In questo caso, infatti, tutti i client sono collegati tra loro in modo peer-to-peer (P2P) per eseguire l'addestramento del modello, come mostrato nella figura 1.2 di seguito.

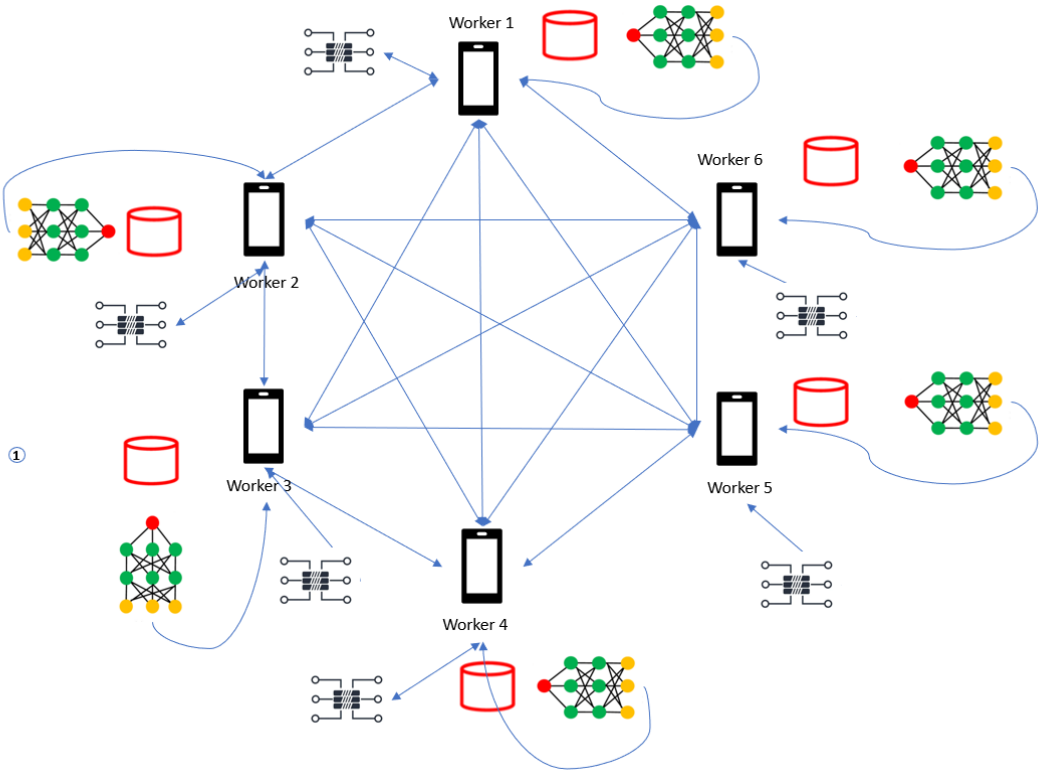

Figura 1.2: Federated learning decentralizzato.

In questo modo, ad ogni iterazione, i client eseguono l'addestramento locale basato sul proprio dataset. Successivamente, ogni client invia i risultati ottenuti dall'addestramento in locale agli altri client, da cui allo stesso tempo riceve le informazioni, risultato della loro iterazione. Quindi ogni client esegue un algoritmo di aggregazione (cap 1.4) e riesegue l'addestramento. Questa operazione è poi ripetuta N volte (round). Durante questi N round vengono creati dei vettori di pesi "globali" su ogni client, che vengono a loro volta inviati e ricevuti. L'identificazione del miglior vettore dei pesi, al fine di ottenere la massima accuratezza del modello, può avvenire in diversi modi. Si può per esempio decidere di far addestrare ogni client autonomamente e solo al termine di n round (con n $<$ N) stabilire il vettore migliore, oppure si può addestrare ogni modello separatamente ed al fine degli N round eleggere il migliore. Una volta identificato il i migliori pesi per il modello questi vengono trasmesso agli altri client, in modo tale che tutti siano allineati e possano continuare l'addestramento con il miglior modello globale.

Il DFL è progettato per sostituire completamente o parzialmente il CFL quando la comunicazione con il server non è disponibile o la topologia di rete è altamente scalabile.
Vantaggi:

- **Privacy e sicurezza:** DFL consente ai nodi periferici di mantenere i propri dati localmente, riducendo il rischio di esposizione dei dati sensibili.
- **Scalabilità:** Poiché la comunicazione avviene direttamente tra i nodi periferici, DFL può essere più scalabile rispetto a CFL, specialmente in scenari con un gran numero di nodi.
- **Ridotta dipendenza da un server centrale:** DFL elimina il singolo punto di fallimento del server centrale, consentendo una maggiore resilienza e autonomia dei nodi periferici.

Svantaggi:

- **Complessità di implementazione:** DFL richiede una maggiore complessità nell'organizzazione e nella sincronizzazione delle comunicazioni tra i nodi periferici, poiché non c'è un server centrale che svolga questo ruolo.
- **Coordinazione tra nodi periferici:** La collaborazione diretta tra i nodi periferici può richiedere una gestione più sofisticata per garantire la coerenza e l'accuratezza del modello globale.

Il DFL è sicuramente una delle architetture di FL più interessanti, il suo punto di forza è la scalabilità, in quanto è possibile realizzare un modello di questo tipo con un numero elevato di partecipanti all'apprendimento federato, questo permette di superare alcuni limiti del FL ((cap 1.3)) ma allo stesso tempo indebolisce le sue prestazioni per alcuni di questi.

1.3 Limitazioni del federated learning

In questo paragrafo affronteremo i principali limiti dell'apprendimento federato (FL), distinguendo su quale delle due architetture menzionate in precedenza (CFL E DFL) questi limiti sono maggiormente influenti. Verranno inoltre menzionate alcune possibili soluzioni o tentativi per superare tali limitazioni.

1.3.1 Efficienza di comunicazione e gestione di sistemi eterogenei

Una sfida attualmente in auge nell'ambito del FL riguarda i costi di comunicazione, dal momento che il FL si svolge in un ambiente distribuito in cui i dispositivi connessi in rete devono condividere costantemente i loro aggiornamenti, questo può creare un collo di bottiglia nella comunicazione(14).
L'efficienza di comunicazione rappresenta un aspetto cruciale che richiede l'utilizzo di metodi di trasmissione ottimali. Ovviamente la mole di comunicazioni tra server e client (CFL) e tra i diversi client(DFL) aumenta in relazione all'aumentare del numero di nodi. Una possibile soluzione potrebbe essere quella di ridurre il numero di nodi partecipanti all'apprendimento federato, tuttavia questo potrebbe comportare una riduzione in termini di precisione del modello globale, infatti, riducendo il numero di nodi si ridurrebbe il numero di campioni su cui il modello viene addestrato. É per questa ragione necessario tentare di individuare un numero di nodi idoneo, al fine di garantire uno scambio di informazioni efficiente ed al contempo un quantitativo di campioni sufficiente per addestrare al meglio il modello.
Nell'articolo intitolato "Client selection for federated learning with heterogeneous resources in mobile edge," (15) gli autori T. Nishio e R. Yonetani propongono una strategia per la selezione dei nodi che parteciperanno all'addestramento federato. Questa strategia si basa sull'idea di inviare una richiesta dal server a tutti i client, il che è possibile solo nell'ambito dell'apprendimento federato centralizzato (CFL). I client devono fornire informazioni sulle proprie risorse, come ad esempio la larghezza di banda di comunicazione o lo stato della batteria. L'approccio utilizzato da Nishio e Yonetani mira ad ottenre questo tipo di informazioni sui diversi host, in quanto i dispositivi partecipanti potrebbero non sempre disporre della larghezza di banda necessaria e potrebbero partecipare in condizioni di rete non affidabili. Sulla base di queste informazioni ricevute, il server seleziona i nodi che possono partecipare all'apprendimento federato.
Questa soluzione a sua volta potrebbe implicare comunque alcuni aspetti negativi, ad esempio potrebbero essere necessari più round di addestramento nel caso di utilizzo di modelli complessi, e la selezione di un numero troppo basso di partecipanti potrebbe portare a una scarsa performance, senza trascurare l'ipotesi che alcuni partecipanti po-

trebbero abbandonare durante l'addestramento. Inoltre, c'è un'inclinazione verso la selezione di partecipanti con dispositivi che dispongono di un quantitativo di risorse superiori, trascurando però il fatto che questi potrebbero non possedere dati rappresentativi della distribuzione della popolazione.
Questo ci porta a trattare un altro limite dei sistemi che basano il loro addestramento sulla partecipazione spontanea dei nodi, ovvero la gestione di sistemi eterogenei. Si rende perciò necessario considerare le differenze in termini di hardware, connettività di rete e alimentazione dei dispositivi presenti nella rete federata. È fondamentale sviluppare metodi in grado di anticipare la scarsa partecipazione dei dispositivi, tollerare l'eterogeneità hardware e resistere alla presenza di dispositivi in shut down nella rete. Per risolvere questo tipo di intoppo si potrebbe ricorrere alla comunicazione asincrona.
La principale differenza tra i sistemi di federated learning basati su comunicazione asincrona rispetto ai sistemi sincroni risiede nel flusso di lavoro. Nei sistemi sincroni, infatti, tutti i partecipanti devono completare le loro operazioni prima di procedere alla fase successiva. Ciò non avviene invece comunicazione asincrona, in cui i partecipanti possono operare in modo indipendente e inviare aggiornamenti al server FL solamente quando sono pronti, senza dover attendere gli altri partecipanti(questa soluzione può essere applicata anche a sistemi decentralizzati). Tuttavia sebbene la comunicazione asincrona possa rappresentate una buona soluzione, è necessario far fronte ad alcune sfide, come la gestione delle versioni del modello, la sincronizzazione dell'aggiornamento dei partecipanti e la gestione degli eventuali conflitti tra gli aggiornamenti ricevuti. Queste sfide richiedono strategie e meccanismi specifici per garantire l'efficienza e la coerenza del processo di apprendimento federato asincrono. L'implementazione di tutti questi controlli rende ben più complessa l'implementazione del sistema federato.

1.3.2 Dati dati non indipendenti e identicamente distribuiti (non-IID)

I dati non indipendenti e identicamente distribuiti (non-IID) sono un tipo di dati in cui i campioni non sono generati da una stessa distribuzione statistica e ciò fa sì che possano essere correlati tra loro. In altre parole, la distribuzione dei dati può variare tra i diversi campioni o gruppi di campioni. Questa condizione può essere generata da fattori come la disomogeneità dei partecipanti, la variabilità temporale o le differenze di contesto.
Nell'ambito del Federated learning è molto probabile che si verifichino casi di addestramento basati su dati non IID, in quanto la probabilità che i dati siano distribuiti in modo identico su tutti i client partecipanti all'addestramento è quasi nulla. In più uno degli aspetti fondamentali del federated learning è proprio la possibilità di addestrare model-

li in sistemi distribuiti in differenti aree geografiche, questo comporta quasi sempre una difformità del contesto in cui vengono raccolti i dati. Per fare un esempio più chiaro, basti pensare ad un sistema di FL che tenta di addestrare un semplice modello di riconoscimento facciale. I client che partecipano all'addestramento sono distribuiti in tutto il mondo di conseguenza ognuno di essi esegue l'addestramento su foto di volti relativi alla popolazione in cui si trova. Questo porta alla creazione di dati non IID in quanto:

- ogni client addestra il modello su un numero arbitrario di immagini che ha a disposizione;
- la differenza tra i volti di individui appartenenti a diverse etnie comporta una forte diversità tra i dati che potrebbero influenzare in maniera negativa l'addestramento del modello globale.

Eśtato dimostrato che l'algoritmo FedAVG (sezione 1.4.2) riesce ad ottenere ancora degli otti risultati in caso di dati non IID (4), ma nell'articolo (7) è riportato che nel caso in cui viene generata una situazione in cui i dati sono distribuiti tra i partecipanti all'addestramento in modo non omogeneo, creando quindi una situazione di non-IID, l'accuratezza delle reti neurali convoluzionali addestrate con l'algoritmo FedAvg può ridursi significativamente, fino al 11% per MNIST, 51% per CIFAR-10 e 55% per i dataset di riconoscimento di parole chiave (KWS).
La gestione dei dati non indipendenti e identicamente distribuiti richiede l'implementazione di strategie specifiche al fine di garantire l'accuratezza e la generalizzazione ottimale dei modelli nel contesto del Federated Learning (FL). L'utilizzo dei dati non IID rappresenta uno dei principali campi di studio e di ricerca nell'ambito del FL, poiché presenta sfide significative. Affrontare la non IID richiede l'adozione di approcci innovativi che tengano conto delle correlazioni e delle variazioni nei dati tra i partecipanti. Ciò può comprendere l'implementazione di algoritmi di aggregazione ponderata, dove viene attribuito maggiore peso ai partecipanti con dati più rappresentativi della popolazione, o l'utilizzo di tecniche di generazione di dati sintetici(Data Augmentation) per creare un insieme di dati bilanciato e più simile alla distribuzione target.
Nel'articolo "Mitigating Data Heterogeneity in Federated Learning with Data Augmentation"(16) è stato dimostrato come grazie all'aumento dei dati, si può riuscire ad ottenere prestazioni all'avanguardia utilizzando anche algoritmi FL più elementari.
Ulteriori studi hanno dimostrato come creando un piccolo database condiviso tra i diversi partecipanti all'addestramento (ovviamente questo va contro uno dei principi cardine del FL, ossia la non condivisione dei dati) si riesce a migliorare di molto l'accuratezza delle predizioni del modello globale. Nell'articolo "Apprendimento federato con dati non IID" (7) gli autori hanno eseguito un test addestramedo un modello secondo la strategia

dell'addestramento federato sul dataset CIFAR-10. Per rendere i dati non IID hanno distribuito i dati assegnando ogni classe ad un singolo partecipante. Hanno poi appurato che condividendo tra i diversi partecipanti all'addestramento il 5% dei dati presenti su ogni client, l'accuratezza del modello può essere aumentata fino al 30%.

1.3.3 Problematiche di privacy

Come spesso sottolineato, data la sua importanza, un punto cardine del FL tratta la privacy dei dati.
Nonostante si tratti di un punto focale, il federated learning presenta comunque diverse problematiche sotto questo aspetto, poiché sono coinvolte la condivisione e l'elaborazione dei dati utente tra dispositivi distribuiti. Alcune delle principali problematiche di privacy associate al federated learning possono essere:

- **Inference attacks (17)(Attacchi di inferenza):** un attaccante potrebbe sfruttare le informazioni apprese dal modello federato per estrarre informazioni specifiche sugli utenti o sui dati di addestramento. L'analisi delle predizioni del modello potrebbe quindi rivelare informazioni personali sugli utenti.

- **Centralizzazione dei dati sensibili:** sebbene il federated learning miri a mantenere i dati utente sui dispositivi locali, è comunque necessaria una certa forma di centralizzazione per coordinare il processo di addestramento e di aggregazione dei modelli. Questo potrebbe comportare rischi di sicurezza e potenziali violazioni dei dati sensibili durante la trasmissione o lo storage centralizzato(nel server o nei diversi client). Questo aspetto è molto più accentuato nel caso in cui ci si trovi nel contesto del DFL, in quanto ogni partecipate potrebbe avere accesso al risultato dell'addestramento degli altri client.

Per risolvere questi problemi è possibile applicare dei metodi che permettono di preservare la riservatezza dei dati durante il processo di apprendimento. L'utilizzo della crittografia omomorfica(18), ad esempio, potrebbe essere una plausibile soluzione dal momento che consente il calcolo di una funzione su input crittografati e produce a sua volta il risultato in forma crittografata. Ad ogni modo esistono anche altre tecniche per garantire la sicurezza dal dato, e ció contribuisce a migliorare la privacy nell'apprendimento federato, sebbene spesso comporti un compromesso in termini di prestazioni ed efficienza del sistema. Per tali ragioni è necessario trovare un equilibrio tra privacy e accuratezza del modello, considerando anche la computazionalità, l'efficienza comunicativa e la tolleranza ai guasti dei metodi di tutela della privacy.

1.3.4 Byzantine attacks

Gli attacchi Byzantine, noti anche come guasti Byzantine, si riferiscono a comportamenti maliziosi o arbitrari esibiti dai partecipanti in un sistema distribuito. Questi attacchi prendono il nome dal cosí detto "Problema dei Generali Bizantini". Si tratta di uno scenario teorico in cui un gruppo di generali deve raggiungere il consenso su un'azione militare coordinata, nonostante vi sia la presenza di generali traditori che potrebbero inviare messaggi contrastanti o ingannevoli. Gli attacchi Byzantine possono manifestarsi in vari modi, ad esempio come nodi che inviano informazioni contrastanti, forniscono dati errati, rifiutano di cooperare o cercano deliberatamente di interrompere il processo di consenso o di presa di decisioni. Tali attacchi possono avere un impatto significativo sull'integrità, la disponibilità e l'affidabilità del sistema distribuito, quindi anche su applicazioni che tentano di addestrare modelli sttraverso l'utilizzo del federated learning.

Nel contesto del federated learning, gli attacchi Byzantine possono verificarsi quando i client partecipanti forniscono intenzionalmente aggiornamenti errati al modello globale durante il processo di aggregazione. Questi attacchi mirano a corrompere il modello aggregato o a indurre in errore il processo di apprendimento. Gli schemi e gli algoritmi di aggregazione resilienti ai Byzantine attacks sono progettati per rilevare e mitigare gli effetti di tali attacchi, garantendo l'integrità e l'affidabilità del modello globale nonostante la presenza di partecipanti maliziosi. Nell'articolo "A Byzantine-Resilient Aggregation Scheme for Federated Learning via Matrix Autoregression on Client Updates" 19 gli autori propongono un nuovo algoritmo di aggregazzione chiamato FLANDERS. Il FLANDERS si fonda su alcune serie temporali che vengono create sulla base dei diversi aggiornamenti che ogni client, appartenente all'addestramento federato, condivide. L'algoritmo in questione, ogni qualvolta che si appresta a eseguire il passaggio di aggregazione dei vari modelli per la costituzione del nuovo modello globale, esegue delle verifiche su serie temporali formate in seguito alla ricezione dei modelli da ogni client. Controllando poi le serie temporali, riesce ad identificare eventuali discostamenti dei dati che gli permettono di individuare nodi malevoli. Come riportato nell'articolo suddetto, questo algoritmo sembrerebbe funzionare in maniera ottimale anche se il numero di nodi malevoli corrisponde al 50% dei partecipanti all'apprendimento federato.

1.4 Algoritmi di aggregazione per il Federate Learmimg

Nelle precedenti sezioni del capitolo 1 abbiamo analizzato i vantaggi e svantaggi dell'apprendimento federato, le diverse topologie che esso prevede e il suo funzionamento. Abbiamo però tralasciato l'argomento dell'aggregazione che viene fatta sui risultati ottenuti da tutti i partecipanti all'apprendimento federetao per creare il modello globale. In questo paragrafo verranno presentati due degli algoritmi maggiormente diffusi, ovvero FedSGD e FedAVG, e le scelte che ci hanno portato all'utilizzo del secondo nei nostri test riportati nel capitolo 5.

1.4.1 FedSGD

L'algoritmo FedSGV, ovvero federated stochastic gradient descent, è basato sulle varianti della discesa del gradiente stocatico(SGD), da cui di fatto questo algoritmo prende il nome.
Questo tipo di metodologia si basa sull'idea di costruire algoritmi per SGD che possono essere applicati al problema dell'ottimizzazione federata.
Uno dei primi approcci è stato quello di eseguire l'algoritmo SGD in modo sincrono, in cui viene eseguito un singolo calcolo del gradiente batch (ad esempio su un client selezionato casualmente) per ogni round di comunicazione. Questo metodo è computazionalmente efficiente, ma richiede un numero molto elevato di round di addestramento per produrre modelli validi. Un esempio è quello riportato nell'articolo di Ioffe e Szegedy (5)], i quali hanno tentato di addestrare un modello che sfruttasse quest'algoritmo sul dataset MINIST.
Nell'ambito federato, non è necessario sostenere un grande costo in termini di tempo effettivo per coinvolgere più client, ragion per cui vi sono stati diversi studi, mediante i quali è stato dimostrato che l'utilizzo di batch più grandi calcolati in maniera sincrona permette di raggiungere una convergenza con meno round e con uno sforzo computazionale anche minore(6)].

Prima di fornire una definizione formale del FedSGV bisogna definire alcuni termini:
Per applicare questo approccio all'ambito federato, selezioniamo una frazione C di client in ogni round e calcoliamo il gradiente su tutti i dati detenuti da questi client. Pertanto, C controlla la dimensione globale del batch, con $C = 1$ che corrisponde alla discesa del gradiente in full-batch (non stocastica).

Una tipica implementazione di FedSGD con $C = 1$ e un tasso di apprendimento fisso η fa sì che ogni client k calcoli

$$g_k = \nabla F_k(w_t)$$

che è il gradiente medio sui dati locali nel modello calcolato dal client w_t. Il server centrale aggrega questi gradienti e applica l'aggiornamento:

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$$

poiché:

$$\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f_i(w_t)$$

Un aggiornamento equivalente è dato da :

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k$$

poi quindi:

$$w_{t+1} \leftarrow \sum_{k=1}^{k} \frac{n_k}{n} w_{t+1}^k$$

In altre parole, ogni client esegue localmente un passo di discesa del gradiente sul modello utilizzando i suoi dati locali, e il server successivamente esegue una media ponderata dei dati restituiti dall'addestramento dei modelli su un gruppo di partecipante all'addestramento federato. Una volta scritto l'algoritmo in questo modo, possiamo aggiungere più calcoli a ogni client iterando l'aggiornamento locale

$$w_k \leftarrow w_k - \eta \nabla f_k(w^k)$$

più volte prima del calcolo della media dei parametri dei diversi modelli locali.

1.4.2 FedAVG

L'algoritmo FedAvg (Federated Averaging) è una generalizzazione del FedSGD. Si tratta di un algoritmo di apprendimento federato che opera in iterazioni multiple e combina l'addestramento locale dei partecipanti con una fase di aggregazione centralizzata. Questo algoritmo si focalizza sui pesi risultati dall'addestramento di ogni modello in locale.
La sua formulazione può essere descritta come segue:

1. **Inizializzazione:** un modello globale è inizializzato al server centrale.
2. **Iterazioni:**
   (a) il server invia il modello globale corrente a tutti i partecipanti;
   (b) ogni partecipante i addestra il modello utilizzando i propri dati locali D_i per un numero prestabilito di epoche o passi di addestramento;
   (c) al termine dell'addestramento i partecipanti inviano i pesi calcolati al server centrale;
   (d) il server aggrega i pesi ricevuti da tutti i partecipanti e calcolando una media pesata dei pesi;
   (e) il server utilizza i pesi aggregati per aggiornare il modello globale;
   (f) reinvia i pesi ai diversi partecipanti all'addestramento federato in modo tale che possano rieseguire l'addestramento con i parametri aggiornati.
3. **Ripetizioni:** è necessario ripetere le iterazioni fino a quando non viene raggiunto un criterio di terminazione predefinito (ad esempio un numero massimo di iterazioni o un valore di perdita desiderato).

L'algoritmo FedAvg utilizza il concetto di media pesata dei pesi per aggiornare il modello globale, in modo che questi rifletta l'informazione aggregata dai partecipanti, senza mai avere accesso diretto ai dati locali. Questo consente di ottenere un modello globale migliore che rappresenta l'informazione distribuita tra tutti i partecipanti.

Definizione formale di FedAvg: Supponiamo di avere un insieme di N dispositivi edge o clienti (denotati da $i = 1, 2, ..., N$) che partecipano al processo di apprendimento federato. Ciascun cliente ha un set di dati locale D_i, non condivisi con gli altri clienti. L'obiettivo è addestrare un modello di machine learning globale utilizzando il paradigma del machine learning federato.

Inizializzazione:

- Inizializza il modello globale con pesi w_0.

- Inizializza il modello aggregato w_{agg} con pesi w_0.

Iterazioni di apprendimento: Per ogni round di addestramento $t = 1, 2, ..., T$:

- Seleziona un sottoinsieme di clienti $C_t \subseteq 1, 2, ..., N$ in modo casuale o secondo una qualche strategia di selezione.
- Per ogni cliente $i \in C_t$:
  - Trasmetti il modello globale w_{agg} a i.
  - Addestra il modello locale di i utilizzando il set di dati locale D_i e il modello w_{agg} per ottenere i nuovi pesi w_i.
  - Trasmetti i pesi del modello locale w_i a un server centrale.
- Aggregazione dei pesi:
  - Calcola il modello aggregato come la media dei pesi dei clienti:

$$w_{\text{agg}} = \frac{1}{C_t} \sum_{i \in C_t} w_i$$

.

Restituzione del modello globale: Restituisci il modello globale w_{agg} ad ogni cliente ripeto l'addestramento per un numero N di volte o finchè non raggiungo un'accuratezza desiderata.

Di seguito riporto lo pseudocodice dell'algoritmo per rendere più chiaro il funzionamento.

Server: Server executes:
Initialize w_0; **for** *each round* $t = 1, 2, \ldots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$; $S_t \leftarrow$ (random set of m clients); **for** *each client* $k \in S_t$ *in parallel* **do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$;
  end
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$;
end

Client: ClientUpdate(k, w): Run on client k
$B \leftarrow$ (split P_k into batches of size B); **for** *each local epoch i from 1 to E* **do**
  for *each batch* $b \in B$ **do**
    $w \leftarrow w - \eta \nabla O(w; b)$;
  end
end
return w to server;

1.5 Frameworck

Esistono differenti Framework che permettono di sviluppare sistemi federati in maniera semplice e agevole. Tre tra i più famosi e utilizzati sono :

1. **TensorFlow Federated (TFF):** (25) È un framework open-source sviluppato da Google che permette di eseguire il Federated Learning utilizzando la libreria TensorFlow. Fornisce un'ampia gamma di strumenti e funzionalità per lo sviluppo di modelli di apprendimento federato.

   - Vantaggi:
     - Integrazione diretta con TensorFlow, consentendo di utilizzare l'ecosistema esistente di TensorFlow per lo sviluppo di modelli di apprendimento federato.
     - Ampia gamma di strumenti e funzionalità per il supporto allo sviluppo di modelli di apprendimento federato.
   - Svantaggi:
     - Documentazione non molto completa , a causa della relativa novità del progetto.
     - Risulta abbastanza complicato da utilizzare nel caso in cui non si abbia familiarità con il framework tensorflow.

2. **PySyft:** (24) È una libreria open-source basata su PyTorch che permette di eseguire il Federated Learning e altre operazioni di Machine Learning sicure e private. PySyft supporta la federazione di modelli tra diversi dispositivi e nodi.

   - Vantaggi:
     - Basato su PyTorch, che è una popolare libreria di deep learning, fornendo un'interfaccia familiare per gli sviluppatori di PyTorch.
     - Permette di creare sistemi federati in maniera semplice e veloce. Concede anche di eseguire delle simulazioni per verificare il funzionamento della federazione.
   - Svantaggi:
     - Può richiedere un po' di tempo per comprendere completamente i concetti di sicurezza e privacy implementati da PySyft. I metodi e le funzionalità implementate non sempre sono spiegati in maniera esaustiva.
     - Meno strumenti e funzionalità predefinite rispetto ad altri framework. Questo comporta la necessità di creare funzioni in maniera manuale per gestire tutte le operazioni.

3. **Flower:** (23)È un framework open-source sviluppato da Adap, che offre un'implementazione semplice ed efficiente del Federated Learning. Flower permette di distribuire l'addestramento dei modelli su dispositivi edge e di aggregare i risultati in modo federato.

   - Vantaggi:

     Implementazione semplice ed efficiente del Federated Learning. Supporto per la distribuzione dell'addestramento su dispositivi edge.

   - Svantaggi:

     (a) Documentazione non esaustiva.

     (b) Potrebbe risultare complessa da integrare con i framework di deep learning

     (c) gestisce in maniera autonoma tutti gli aspetti dell'apprendimento federato non fornendo allo sviluppatore una visione a 365 gradi di quello che sta accadendo.

     In questa tesi, ho deciso di non utilizzare alcun framework preesistente per il federated learning. Questa scelta mi ha permesso di acquisire una comprensione approfondita del funzionamento del federated learning e di affrontare personalmente tutte le sfide associate. Nel corso della tesi, presenterò come ho implementato il mio framework "SmartFed", caratterizzato da una struttura flessibile che consente di addestrare diversi tipi di modelli di deep learning. Infatti, la struttura è stata progettata in modo tale che sostituendo una singola riga di codice sorgente, sia possibile addestrare qualsiasi tipo di modello di deep learning. In più risulta molto semplice modificare il codice sorgente per aggiungere ulteriore logica al processo di addestramento federato, in modo da renderlo ad esempio più sicuro o efficace.

Capitolo 2

YOLO (You Only Look Once)

2.1 Introduzione a YOLO

Nella stragrande maggioranza dei casi i migliori riconoscitori di immagini ed oggetti sono gli esseri umani, ma come sostiene il premio Nobel Herbert Simon la cui scoperta principale è che il riconoscimento dei modelli è fondamentale nella maggior parte delle attività decisionali umane: "Più modelli rilevanti avete a disposizione, migliori saranno le vostre decisioni (8) . Questa stessa affermazione può essere traslata all'ambito informatico, soprattutto se i temi in questione sono neural networks ed image recognition.
Una rete neurale artificiale (ANN) è un paradigma di elaborazione delle informazioni che si ispira al modo in cui i sistemi nervosi biologici, come il cervello, elaborano le informazioni. Le caratteristiche principali delle reti neurali sono la capacità di apprendere complesse relazioni non lineari tra input e output, di utilizzare procedure di addestramento sequenziali e di adattarsi ai dati. La famiglia di reti neurali più comunemente utilizzata per i compiti di classificazione dei modelli è quella delle reti feed-forward, che comprende i perceptron multistrato. Ovviamente come suddetto, affinché una rete neurale possa funzionare a dovere deve essere addestrata, deve quindi apprendere da alcuni dati per poter avviare un ponderato processo decisionale.
Il processo di apprendimento prevede l'aggiornamento dell'architettura della rete e dei pesi di connessione, in modo che la rete possa eseguire in modo efficiente uno specifico compito di classificazione/clusterizzazione. Una ANN viene configurata per un'applicazione specifica, come il riconoscimento di immagini o la classificazione di dati, attraverso un processo di apprendimento, non viene programmata per risolvere un dato problema ma viene addestrata mediante una serie di esempi della realtà da modellare.
Fanno parte delle attività salienti che possono essere svolte dalle reti neurali il rilevamento e la localizzazione di oggetti nelle immagini. Questo processo identificativo prende il nome di image recognition e viene introdotto nel 2012 quando Alexnet, un modello

di CNN (convolutional neural network) progettata dai dottorandi Ilya Sutskever e Alex Krizhevsky, vince l'ILSVRC (ImageNet Large Scale Visual Recognition Competition).
Una semplice rete neurale, infatti, non è ottimizzata per poter effettuare il riconoscimento di immagini, si avrebbero un elevato carico computazionale ed una scarsa accuratezza, proprio per questa ragione sono state implementate le reti neurali convolute o CNNs (convolutional neural networks). Queste particolari reti neurali ad hoc per l'image recognition traggono il vantaggio dal fatto che in una stessa immagine la vicinanza tra pixel è altamente correlata alla somiglianza, limitando così le connessioni.
In quale modo quindi una rete neurale procede per identificare un oggetto o altro all'interno di un'immagine? Prima di tutto il primo strato si occupa del rilevamento di bordi, linee e cambi di luminosità. Le informazioni raccolte vengono poi passate al secondo livello che si occupa dell'identificazione delle forme. Si prosegue quindi di livello in livello, fino a quando il livello più profondo sarà in grado di rilevare gli oggetti specifici, accorpando tutte le caratteristiche raccolte dagli altri livelli e producendo una o più previsioni di classificazione degli oggetti identificati. La previsione verrà poi confrontata con il dato relativo all'effettiva classe dell'immagine, secondo quanto appreso nella fase di addestramento. Gli ''scarti", o valori previsti errati, verranno riutilizzati per riaddestrare la rete e migliorarne l'accuratezza. Procedendo in questo modo, e con molteplici addestramenti, si migliora via via l'accuratezza della rete, al fine di ridurre l'errore e raggiungere risultati soddisfacenti. Nel mio elaborato ho deciso di utilizzare uno dei modelli di image recognition più famosi al modo YOLO (You Only Look Once). Questo algoritmo non si occupa solo del riconoscimento di oggetti ma è in grado di rilevarli all'interno dell'immagine eseguendo una solo passaggio dell'immagine.
Il rilevamento degli oggetti all'interno di un'immagine può avvenire principalmente in due modi:

- **A scatto singolo,** ovvero utilizza un singolo passaggio dell'immagine di ingresso per fare previsioni sulla presenza e sulla posizione degli oggetti nell'immagine. L'elaborazione di un'intera immagine in un unico passaggio, rende questa metodologia efficiente dal punto di vista computazionale, ma meno accurata nel rilevamento di oggetti di piccole dimensioni. Questi tipi di algoritmi sono adatti per rilevare oggetti in tempo reale in ambienti con risorse limitate;

- **A due scatti,** in questo caso il rilevamento di oggetti avviene eseguendo due passaggi dell'immagine di ingresso per fare previsioni sulla presenza e sulla posizione degli oggetti. Il primo passaggio viene utilizzato per generare una serie di proposte o posizioni potenziali degli oggetti, mentre il secondo passaggio viene utilizzato per affinare queste proposte e fare le previsioni finali. Questo approccio è più accurato del precedente, ma è anche più costoso dal punto di vista computazionale,

infatti, questi algoritmi sono utilizzati nei casi in cui risulta prevalere la precisione, nell'individuazione di oggetti anche di piccole dimensioni, sull'efficienza.

Un popolare modello di rilevamento degli oggetti noto per la sua velocità e precisione è YOLO, acronimo di You Only Look Once.

2.2 Architettura YOLO

Al giorno d'oggi le esigenze vanno molto oltre la semplice classificazione o localizzazione di oggetti in immagini statiche, la necessità è sì quella di classificare e localizzare, ma soprattutto di avere queste analisi in tempo reale. Si pensi ad esempio alle auto driverless (ad esempio Tesla), chi vorrebbe trovarsi a bordo di un'automobile autonoma che impiega vari minuti o anche solo diversi secondi per effettuare il riconoscimento? Chi vorrebbe avere una sorveglianza con tempi molto dilatati? È da considerare oltretutto che esistono difficoltà non banali anche in immagini statiche, lo stesso oggetto può assumere forme differenti e presentarsi secondo diversi orientamenti.
La soluzione al problema è quella di utilizzare reti convolute a passata singola, ovvero che analizzano tutte le parti dell'immagine in parallelo, simultaneamente, consentendo di evitare l'uso della sliding window.
YOLO è un rilevatore a scatto singolo che utilizza una rete neurale completamente convoluzionale (CNN) per elaborare un'immagine. È stato introdotto per la prima volta da Joseph Redmon et al. nel 2016 (20), e da allora ha subito diverse iterazioni, l'ultima delle quali è YOLO v8, sebbene nel presente lavoro di tesi sia stata utilizzata la v7.
YOLO è sviluppato su Darknet che è un framework open source per reti neurali scritto in C e CUDA. È veloce e supporta sia il calcolo su CPU che su GPU. Darknet si installa con solo due dipendenze opzionali: OpenCV se gli utenti desiderano una più ampia varietà di classi di immagini supportate, CUDA se invece vogliono il GPU Computing. È un framework veloce e altamente accurato (l'accuratezza di un modello di addestramento personalizzato dipende dai dati di addestramento, dalle epoche, dalla dimensione del batch e da alcuni altri fattori) per il rilevamento di oggetti in tempo reale (può essere utilizzato anche per le immagini).
La caratteristica principale di YOLO v7 è la sua incredibile velocità di identificazione, è infatti in grado di elaborare le immagini a una velocità di 155 fotogrammi al secondo, rispetto ai 45 della sua prima versione. Pecca però di accuratezza rispetto ad altre reti, in particolar modo quando si tratta di individuare oggetti di piccole dimensioni. Questo collima con quanto spiegato nel paragrafo precedente, infatti, si tratta di rilevamento a scatto singolo. Tuttavia, nella versione di YOLO v7, si ha un miglioramento fondamentale per il superamento del limite appena citato, infatti:

- viene utilizzata una funzione di ''perdita focale", che rispetto alla funzione di perdita standard utilizzata in tutte le versioni precedenti, permette di ridurre la perdita degli oggetti ben classificati concentrandosi su quelli difficili da rilevare;

- presenta una risoluzione più elevata rispetto alle versioni precedenti che permette di elaborare le immagini a una risoluzione di 640 x 640 pixel, superiore a quella di 416 x 416 utilizzata in YOLO v3. Questa maggiore risoluzione consente a YOLO v7 di rilevare oggetti più piccoli e di avere una maggiore precisione complessiva.

Di seguito un'immagine che rappresenta l'architettura del modello CNN che costituisce il pilastro su cui opera YOLO:

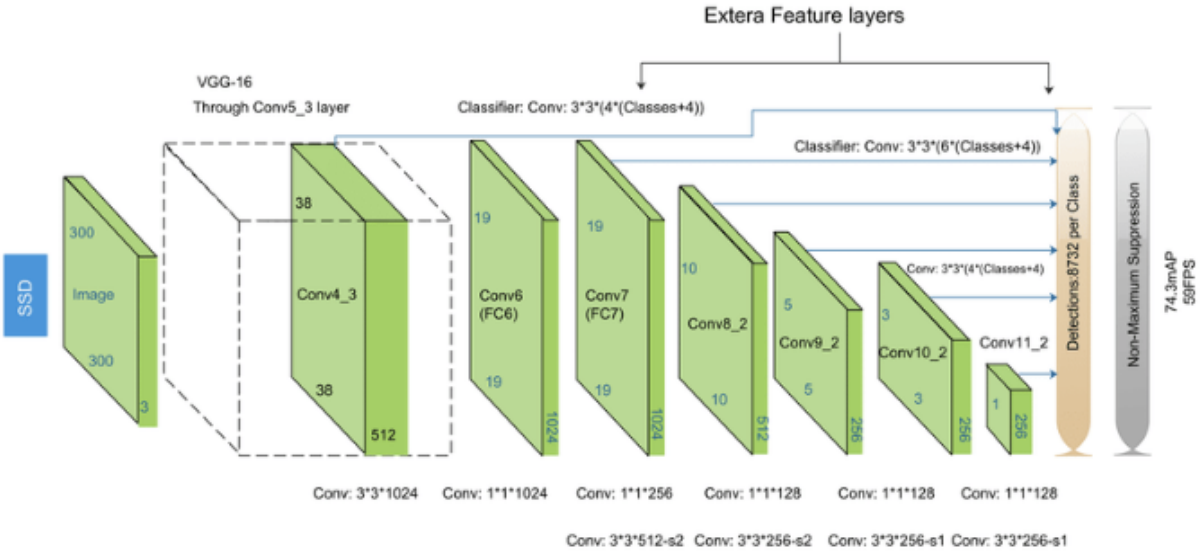

Figura 2.1: Esempio di CNN con architettura SSD.

Per semplificare come vengono analizzate le immagini da YOLO facciamo una breve analisi step by step:

1. **Divisione dell'immagine:** l'immagine di input viene suddivisa in una griglia di celle chiamate anchor boxes o caselle di ancoraggio, dove ogni cella rappresenta una regione potenziale in cui potrebbe trovarsi un oggetto. YOLO v7 utilizza nove caselle di ancoraggio, il che consente di rilevare una gamma più ampia di forme e dimensioni di oggetti rispetto alle versioni precedenti, contribuendo così a ridurre il numero di falsi positivi;

2. **Estrazione delle caratteristiche:** utilizzando una rete neurale convoluzionale (CNN), vengono estratte le caratteristiche significative dall'immagine (luci, ombre, bordi, pixel...). Queste caratteristiche consentono di comprendere la struttura dell'immagine e di individuare gli oggetti;

3. **Predizione:** per ognuna delle nove celle della griglia, YOLO predice le coordinate delle bounding box (ovvero i rettangoli che delimitano gli oggetti identificati) e le probabilità di appartenenza alle diverse classi di oggetti. Ogni cella può contenere e predire un insieme di bounding box e classi;

4. **Scrematura delle previsioni:** non tutte le previsioni effettuate presentano la stessa probabilità, ragion per cui vengono scartate tutte quelle che non superano la una soglia di confidenza desiderata. In altre parole, solo le previsioni con una probabilità superiore alla soglia di confidenza vengono considerate come rilevamenti validi. La scelta specifica della soglia di confidenza dipende dal contesto e dai requisiti dell'applicazione, in genere una soglia più alta comporta una maggiore precisione ma potrebbe perdere alcuni falsi negativi, mentre una soglia più bassa potrebbe includere più falsi positivi. Tali soglie di valutazione saranno approfondite in seguito nel paragrafo 2.4;

5. **NMS (Non-Maximum Suppression):** per evitare sovrapposizioni ridondanti di bounding box per lo stesso oggetto, viene applicata la tecnica di NMS, la quale tiene conto dell'intersezione over unione (IoU) tra le bounding box e prende in considerazione solamente quella con la maggiore probabilità;

6. **Output e previsioni:** le bounding box rimanenti, ovvero quelle con probabilità superiore alla soglia di confidenza prestabilita, insieme alle relative classificazioni, costituiscono l'output del modello YOLO. Queste informazioni vengono utilizzate per rilevare e classificare gli oggetti nell'immagine di input.

Tuttavia è da riconoscere che sebbene YOLO v7 sia un algoritmo di rilevamento degli oggetti potente, efficace e veloce, presenta alcune limitazioni. Come detto in precedenza fatica ad individuare oggetti di piccole dimensioni, soprattutto in immagini affollate o quando l'oggetto si trova in una posizione molto distante. Oltre a questo qualche difficoltà nell'identificazione delle classi potrebbe essere data dalle diverse scale presenti nell'immagine, questo può rendere di non facile identificazione gli oggetti troppo grandi o troppo piccoli. Inoltre, YOLO è sensibile anche al variare della luminosità all'interno di un'immagine. Dal punto di vista computazionale, invece, essendo intensivo potrebbe risultare di difficile esecuzione su smartphone o altri dispositivi edge.

2.3 Utilizzo

Il presente lavoro di tesi si propone come obiettivo quello di testare l'accuratezza di un modello addestrato seguendo i principi del FL, e di effettuare poi un confronto con i risultati ottenuti per lo stesso modello sottoponendolo però un addestramento standard, utilizzando quindi un database centralizzato.
Per le simulazioni ho deciso di utilizzare Yolo v7, al fine di testare le prestazioni dell'algoritmo in un ambiente federato. Nel capitolo 5 saranno presentati ed approfonditi i risultati ottenuti dall' addestramento appena introdotto. É ora necessario fare alcune premesse sui motivi che mi hanno portato a selezionare questo modello ed a come si adatti al contesto del FL.
In primis è da precisare che vengono messe a disposizione dagli sviluppatori diverse versioni di yolo v7 ognuna con le proprie caratteristiche:

- YOLOv7 è il modello di base ottimizzato per il normale calcolo GPU.

- YOLOv7-tiny è un modello base ottimizzato per GPU edge. Il suffisso "tiny" dei modelli di visione artificiale indica che sono ottimizzati per i carichi di lavoro Edge AI e deep learning più leggeri per eseguire il processo di machine learning su dispositivi di elaborazione mobile , server e dispositivi edge distribuiti. Questo modello è importante per le applicazioni di visione artificiale distribuite nel mondo reale. Rispetto alle altre versioni, YOLOv7-tiny utilizza Leaky ReLU come funzione di attivazione, la quale fa sì che la rete possa avere un gradiente anche in quelle regioni in cui il ReLU ordinario avrebbe azzerato tutti i valori (fig:2.2)

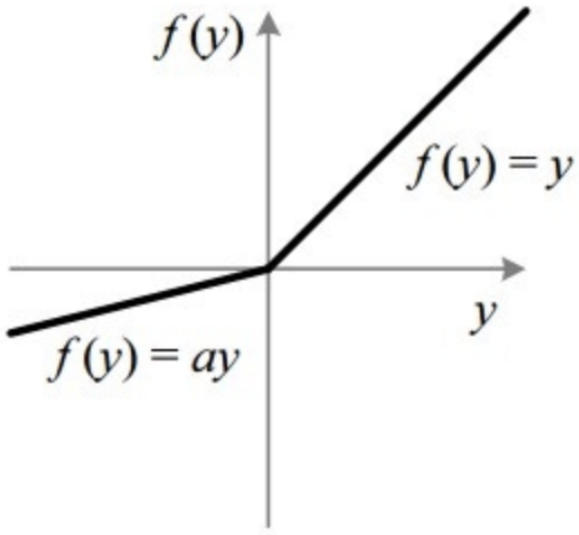

Figura 2.2: LeakyReLU

- YOLOv7-W6 è un modello di base ottimizzato per il cloud GPU computing. Tali unità grafiche cloud (GPU) sono istanze di computer per l'esecuzione di applicazioni in grado di gestire enormi carichi di lavoro di intelligenza artificiale e deep learning nel cloud, senza richiedere l'implementazione delle GPU sul dispositivo dell'utente locale.

Per i miei test, ho scelto di utilizzare la versione YOLOv7-tiny in quanto si adatta meglio a un contesto di Federated Learning (FL), in cui l'addestramento potrebbe essere eseguito su dispositivi di vario tipo. Inoltre, al fine di accelerare l'addestramento, ho deciso di utilizzare un modello preaddestrato sul dataset COCO. Questo mi ha permesso di sfruttare il processo di Transfer Learning (21), il quale consente di riaddestrare modelli già preaddestrati su dataset di grandi dimensioni, adattandoli al contesto specifico del mio dataset di interesse (Vis-DroneDataset). IL processo di trasfert learning si basa su due punti fondamentali :

1. **Rimozione del livello di output:** rimuovendo il livello di output del modello preaddestrato è possibile mantenere solo la parte convoluzionale che estrae le caratteristiche delle immagini. Questo è plausibile poiché il livello di output è specifico del compito di classificazione e determina l'appartenenza alle diverse classi degli oggetti analizzati. Nel caso del dataset COCO, il modello preaddestrato ha un livello di output che corrisponde a 150 classi diverse. Tuttavia, nel mio dataset di interesse, ho solo 9 classi specifiche.

2. **Aggiunta di un nuovo livello di output:** l'aggiunta di un nuovo livello di output personalizzato al modello, con 9 unità corrispondenti alle mie classi di interesse, fa sì che il modello preaddestrato venga adattato alle specifiche classi degli oggetti nel mio dataset.

2.4 Metriche di valutazione

Ovviamente è necessario valutare le prestazioni predittive di rilevamento ed identificazione degli oggetti dei modelli mediante delle metriche di valutazione standard. Le due metriche di valutazione più comuni sono:

- **IoU:** acronimo di Intersection over Union, è una metrica adottata per misurare l'accuratezza della localizzazione e calcolare gli errori di localizzazione nei modelli di rilevamento degli oggetti. L' Intersection over Union è data dal rapporto tra l'area di intersezione e l'area di unione tra i riquadri di delimitazione previsti dal modello e quelli reali dell'oggetto che si vuole identificare nell'immagine. L'intersezione divisa per l'unione fornisce il rapporto tra la sovrapposizione e l'area totale, fornendo una buona stima di quanto il rettangolo di selezione previsto sia vicino al rettangolo di selezione originale. Un valore IoU $>$ 0,5 è considerato una previsione positiva, mentre un valore IoU $<$ 0,5 è una previsione negativa.

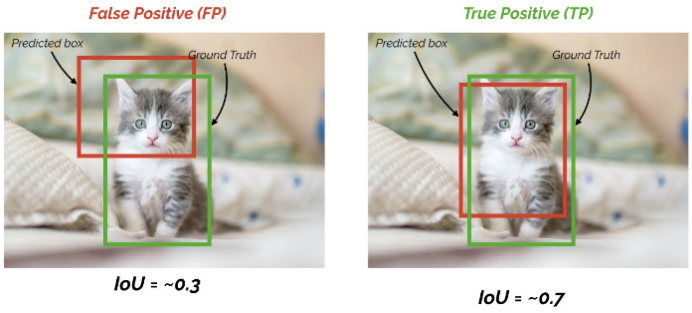

Figura 2.3: Intersection over Union threshold $<$ 0,5 o $>$ 0,5.

- **mAP:** sigla per mean Average Precision, è utilizzata per valutare la precisione complessiva di un modello. Al fine di calcolare la precisione media (mAP) è necessario prima di tutto calcolare la precisione (AP). Per spiegare meglio l'AP (Average Precision), è utile comprendere alcuni concetti chiave di seguito specificati:
  - **Precisione:** questa è data dal rapporto tra le previsioni corrette rispetto e tutte le previsioni positive effettuate dal modello. Si calcola quindi come il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi positivi.

$$\text{Precisione} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  - **Richiamo:** noto anche come recall, misura il rapporto tra gli oggetti positivi correttamente rilevati dal modello ed il numero totale di oggetti positivi presenti nella scena. Si calcola come il rapporto tra i veri positivi e la somma dei

veri positivi e dei falsi negativi.

$$\text{Richiamo} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

– **Curva Precisione-Richiamo:** la curva Precisione-Richiamo viene creata variando la soglia (IoU) di confidenza del modello e calcolando la precisione e il richiamo corrispondenti a ciascuna soglia. Questo permette di valutare come la precisione e il richiamo variano al variare della sensibilità del modello.

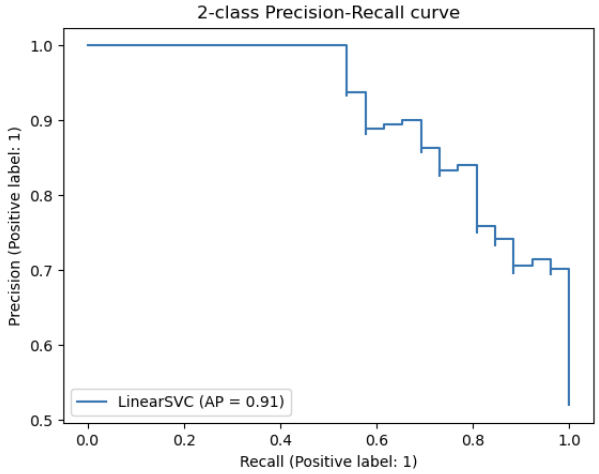

Figura 2.4: Esempio di curva Precisione-Richiamo.

Per determinare l'AP, quindi, si calcola l'area sottesa alla curva Precisione-Richiamo. Questo può essere fatto utilizzando diverse tecniche di interpolazione, ad esempio l'interpolazione di precisione media. L'AP rappresenta quindi la media delle precisioni calcolate per ogni valore di richiamo nella curva. Una volta calcolati i valori di AP per tutte le classi di oggetti di interesse, calcoliamo la media di questi valori per ottenere l'mAP complessivo del modello. L'mAP tiene conto delle prestazioni del modello su tutte le classi, fornendo una valutazione complessiva della sua precisione. È importante notare che l'mAP può essere calcolato utilizzando diverse soglie di IoU (Intersezione over Unione), che determinano la sovrapposizione minima richiesta tra una previsione del modello e una regione di interesse vera per considerarla come una corretta rilevazione. La scelta delle soglie di IoU dipende dal contesto del problema e può variare a seconda delle esigenze specifiche. In sintesi, l'mAP è una metrica utilizzata per valutare l'accuratezza complessiva di un modello di rilevamento degli oggetti, considerando precisione, richiamo e soglie di IoU. La formula della Mean Average Precision (mAP) può essere scritta come:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP(i)$$

Dove:

– mAP rappresenta la Mean Average Precision.

– N è il numero totale di casi di test.

– $AP(i)$ rappresenta l'Average Precision per caso di test i.

La mAP è quindi un compromesso tra precisione e richiamo e considera sia i falsi positivi che i falsi negativi. Questa caratteristica rende la mAP una metrica di valutazione adatta alla maggior parte delle applicazioni di rilevamento delle classi.

- **Confusion Matrix**: nota anche come tabella di contingenza, una confusion matrix è la rappresentazione delle prestazioni di un modello di classificazione su un set di dati. La matrice mostra il numero di predizioni corrette e errate fatte dal modello, organizzate in base alle classi reali e alle classi predette.

  La confusion matrix 2.5 è comunemente utilizzata per valutare le prestazioni di un modello di classificazione in termini di accuratezza, precisione, recall ed altre metriche. È particolarmente utile quando sono coinvolte più classi nel problema di classificazione.

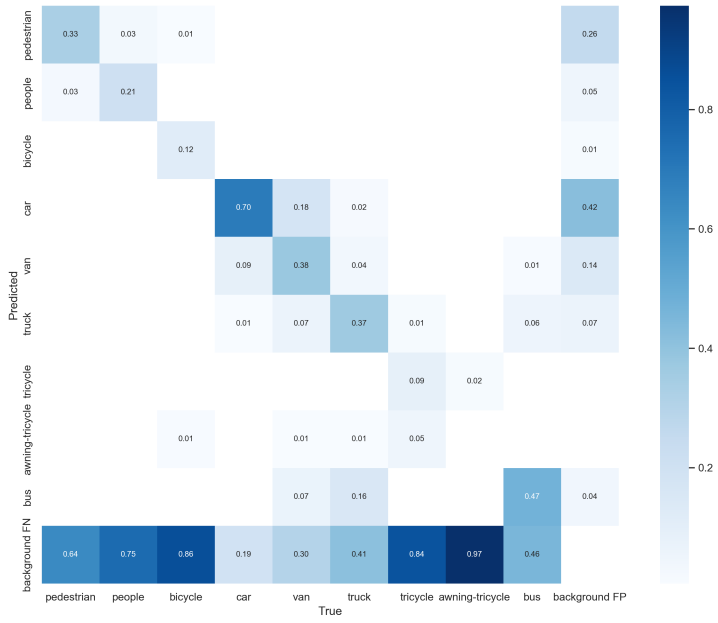

Figura 2.5: confusion_matrix

Mediante le informazioni fornite dalla matrice di confusione possono essere calcolate diverse metriche di valutazione del modello, di seguito alcuni esempi:

– Accuracy (accuratezza): la quale misura la percentuale di predizioni corrette rispetto al totale delle predizioni;

- Precision (precisione): serve per stimare la percentuale di predizioni positive corrette rispetto al totale delle predizioni positive;
- Recall (richiamo): valuta la percentuale di campioni positivi correttamente classificati rispetto al totale dei campioni positivi;
- F1-Score: è la media armonica tra precisione e recall, fornisce una misura complessiva delle prestazioni del modello.

La confusion matrix offre dunque una visione dettagliata delle prestazioni del modello ed aiutare ad identificare specificamente quali tipi di errori vengono commessi. È una valutazione essenziale per comprendere le prestazioni di un modello di classificazione e guidare le successive ottimizzazioni del modello stesso.

Capitolo 3

VisDrone-Dataset

La selezione di un database affidabile e ben strutturato è essenziale per ottenere risultati accurati e significativi. Pertanto, è di fondamentale importanza comprendere la natura e le caratteristiche del database utilizzato nel contesto della ricerca. Con "selezione del database affidabile" non si vuole intendere un database perfettamente strutturato e distribuito, bensì un database che sia il più possibile veritiero e rappresentativo dello scenario che si è deciso di analizzare, escludendo dati che potrebbero danneggiare il processo di addestramento. In questa parte dell'elaborato, illustrerò il database utilizzato come fondamento per l'analisi condotta nel presente lavoro di tesi.

3.1 Introduzione

Per eseguire tutti i miei test e di conseguenza per valutare l'efficienza dell'addestramento federato ho deciso di utilizzare Il dataset VisDrone (22). Questo dataset è stato creato per affrontare le sfide relative al rilevamento ed al riconoscimento di oggetti tramite intelligenza artificiale nelle applicazioni di sorveglianza aerea.
VisDrone è stato sviluppato con l'obiettivo di fornire una vasta collezione di immagini e video aerei per consentire la ricerca e lo sviluppo di algoritmi di rilevamento e tracciamento degli oggetti. Gli autori del dataset hanno raccolto dati da diverse fonti, utilizzando diverse piattaforme di acquisizione come droni, elicotteri e telecamere montate su veicoli. Le acquisizioni sono state effettuate in una varietà di scenari urbani, tra cui centri cittadini, parchi, incroci stradali, porti, stadi ed altri ambienti comuni. Questo dataset comprende chiaramente diverse categorie di oggetti, tra cui persone, veicoli, biciclette, pedoni, autoveicoli, autobus, motociclette e altro ancora. Le immagini del dataset sono corredate da etichette dettagliate, che stanno ad indicare la posizione e la classe degli oggetti presenti. Queste etichette consentono di valutare le prestazioni del modello di ML (YOLO) applicando un addestramento federato.

Nella mia tesi ho deciso di considerare solo il dataset composto da foto scattate dai diversi dispositivi. La decisione di tralasciare quello composto da video è dovuta puramente ai limiti fissati dalla potenza computazionale (hardware) a mia diposizione. Come avremo modo di approfondire nel capitolo (5), è stato raggiunto un discreto livello di accuratezza, ma ciò non esclude che il risultato dell'addestramento potrebbe essere testato su dataset comprendenti video in eventuali studi futuri.
Tornando all'analisi del dataset utilizzato per l'addestramento del mio modello, è essenziale specificare che è suddiviso in 3 set distinti:

1. dataset di addestramento, utilizzato per la fase iniziale di addestramento del modello. Questo dataset è composto da 6471 immagini;
2. dataset di valutazione, utile per valutare i risultati dell'addestramento ad ogni singola epoca, in questo modo si potrebbe procedere con una fase di tuning degli iperparametri(es. lr, deltaMAP ecc.). Questo dataset è composto da 548 immagini;
3. dataset di test, per valutare l'accuratezza del modello sottoponendogli dati che il modello non ha mai "visto" prima. Questo dataset è composto da 1610 immagini.

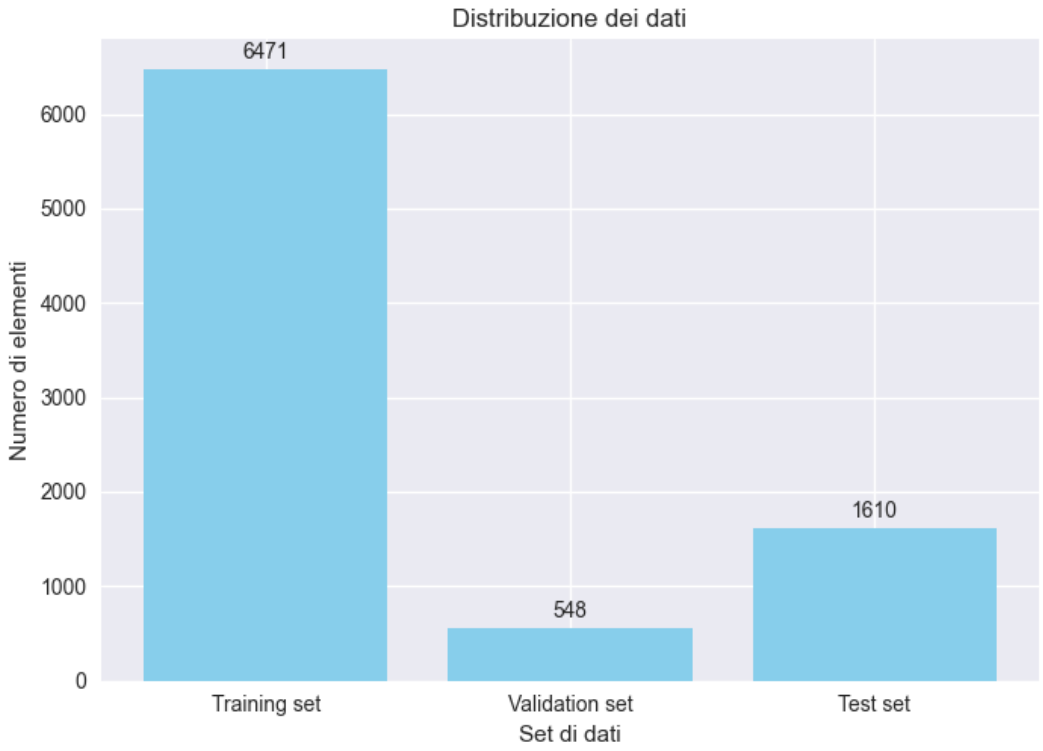

Figura 3.1: Distribuzione dati Trainig Val e Test

Il dataset utilizzato per i miei esperimenti è composto da un totale di 8629 immagine 3.1, ognuna delle quali è accompagnata da un'annotation contenente informazioni sugli

oggetti presenti all'interno di ciascuna immagine. La scelta di utilizzare questo dataset è stata guidata dalle regole fondamentali dell'addestramento federato.
Il dataset VisDrone offre una vasta varietà di immagini catturate da diverse posizioni e con diversi tipi di apparecchiature (aerei, droni, elicotteri ecc.). Nell'ambito dei miei esperimenti, ho voluto esplorare come l'utilizzo di queste diverse fonti di dati possa influenzare l'addestramento di un modello locale, in cui ogni partecipante addestra il proprio modello senza condividere informazioni specifiche sulle proprie immagini. Di fatto è facilmente immaginabile come prima che queste immagini venissero raccolte nel dataset centralizzato (Vis-Drone) fossero salvate su ogni singolo dispositivo che le ha raccolte.
L'obiettivo preposto è dunque quello di valutare come, attraverso l'addestramento federato, e quindi senza che i dati vengano raccolti in un database centralizzato, sia effettivamente possibile ottenere un modello affidabile, in grado di generalizzare bene su dati provenienti da diverse fonti senza compromettere la privacy e la riservatezza delle informazioni. Ovviamente quando si parla di affidabilità si intende dire che l'obiettivo è quello di raggiungere un'accuratezza prossima a quella che si otterrebbe con un addestramento "classico", centralizzando i dati in un singolo dataset. L'utilizzo di un dataset così diversificato ha fornito una base solida per valutare l'efficacia di questa metodologia, oltre che per analizzare come i diversi contributi locali possano concorrere alla creazione di un modello aggregato che si avvicini il più possibile alle prestazioni dello stesso modello addestrato con la metodologia classica.

3.2 Analisi esplorativa Dataset

Come asserito nel paragrafo precedente, il dataset è già suddiviso in 3 set distinti. Dal momento che il mio obiettivo è quello di identificare gli oggetti per poi categorizzarli, al fine di avere un'idea delle label analizzate, non è sufficiente fare riferimento al numero di immagini contenute nei diversi set, ma è necessario analizzare quanti e quali sono gli oggetti racchiusi all'interno dei diversi dataset, e quindi in ciascuna immagine (vedi Figure 3.2, 3.3, 3.4). L'analisi delle distribuzioni dei diversi set di dati rivela un forte sbilanciamento nel database, infatti, le due classi con il numero prevalente di campioni sono costituite da macchine e pedoni, mentre classi come tricicli e biciclette presentano pochissime occorrenze. Questo sbilanciamento porta con se delle conseguenze. Al termine dell'addestramento, infatti, il modello sarà in grado di fare inferenza in maniera molto più precisa sulle classi con un numero di campioni maggiore, poichè avendo più pattern a disposizione per il training sarà in grado di identificare più features utili all'identificazione. Una caratteristica tra tutte che può risultare rilevante è la posizione dell'oggetto. Se ad un modello viene sottoposta l'immagine di una macchina vista da più prospettive, con

colori differenti o diversi modelli, sarà senz'altro in grado di riconoscerla più facilmente rispetto ad una bici vista da un'unica angolazione poichè meno presente nel campione di addestramento.
Prima di procedere con i test, è stato necessario elaborare i dati in diverse modalità. In alcuni casi, ho eseguito esperimenti mirati a sbilanciare ulteriormente i dati sui diversi Worker in modo da creare una situazione di dati non-IID capitolo 1 paragrafo 1.3.2 maggiormente accentuata, mentre in altri casi ho cercato di avvicinarmi a una distribuzione più uniforme per valutare i risultati in una situazione ottimale.

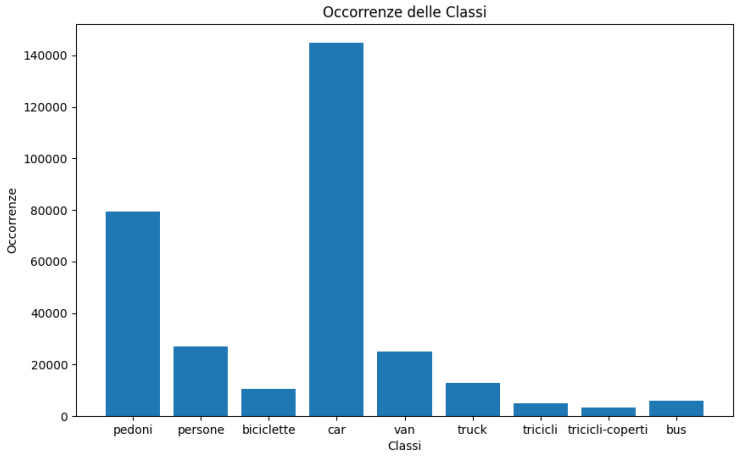

Figura 3.2: Training set

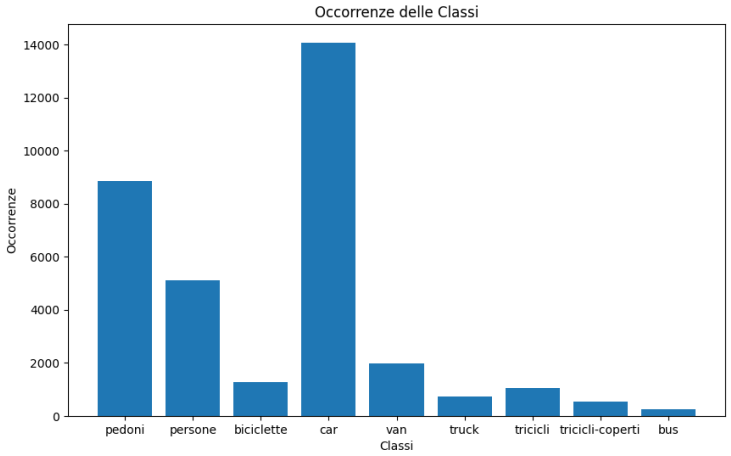

Figura 3.3: Validation set

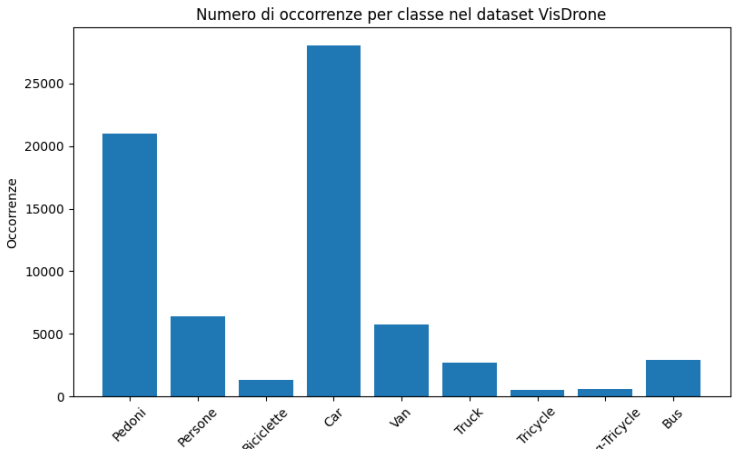

Figura 3.4: Test set

Effettuando una prima analisi delle immagini possiamo notare che presentano una risoluzione molto alta ed un considerevole quantitativo di oggetti al loro interno(3.5).

Figura 3.5: Immagine dataset con Box rappresentati.

Le annotazioni contenute nel dataset forniscono informazioni dettagliate sulla posizione e la classe di ogni oggetto presente nelle immagini. Queste informazioni sono fondamentali per valutare l'accuratezza del modello di rilevamento degli oggetti.
Dopo aver applicato l'algoritmo di tracciamento dei bounding box sugli oggetti rilevati, possiamo confrontare i bounding box predetti dal modello con quelli effettivi riportati nelle annotazioni. Questo confronto ci consente di calcolare una metrica chiamata mAP (mean Average Precision). L'utilizzo delle annotazioni assieme alle valutazione della mAP, permettono di determinare una valutazione quantitativa dell'efficacia del modello di rilevamento degli oggetti, fornendo una misura obiettiva della sua capacità di identificare correttamente gli oggetti e di localizzarli con precisione nelle immagini.
Nelle annotation sono presenti informazioni relative a 8 caratteri utili per identificare l'oggetto come è possibile vedere nella tabella 3.1 Nel nostro caso, utilizzando l'algoritmo YOLO, non è stato necessario valersi di tutte le informazioni dettagliate riportate nella tabella **??**. YOLO, infatti, richiede solo 5 informazioni fondamentali spiegate e riportate nella tabella 3.2, tramite delle opportune trasformazioni è stato possibile estrarre le informazioni necessarie dalle annotazioni presenti nel dataset.
La prima tabella di seguito contiene tutte ed otto le annotation, mentre la seconda presenta solo quelle richieste da YOLO:

Campo	Descrizione
<bbox_left>	La coordinata x dell'angolo in alto a sinistra del bounding box predetto
<bbox_top>	La coordinata y dell'angolo in alto a sinistra del bounding box predetto
<bbox_width>	La larghezza in pixel del bounding box predetto
<bbox_height>	L'altezza in pixel del bounding box predetto
<score>	Lo score nel file DETECTION indica la fiducia del bounding box predetto che racchiude un'istanza dell'oggetto. Lo score nel file GROUNDTRUTH è impostato a 1 o 0. 1 indica che il bounding box viene considerato nella valutazione, mentre 0 indica che il bounding box viene ignorato.
<object_category>	La categoria indica la classe di appartenenza dell'oggetto annotato, (ad esempio regioni ignorate (0), pedoni (1), persone (2), biciclette (3), auto (4), furgoni (5), camion (6), tricicli (7), tricicli con tetto (8), bus (9), altri (10))
<truncation>	Lo score nel file DETECTION deve essere impostato alla costante -1. Lo score nel file GROUNDTRUTH indica il grado in cui le parti dell'oggetto appaiono al di fuori del frame (ad esempio nessuna troncatura = 0 (percentuale di troncatura 0%), troncatura parziale = 1 (percentuale di troncatura 1% 50%)).
<occlusion>	Lo score nel file DETECTION deve essere impostato alla costante -1. Lo score nel file GROUNDTRUTH indica la frazione di oggetti oscurati (ad esempio nessuna oscurazione = 0 (percentuale di oscurazione 0%), oscurazione parziale = 1 (percentuale di oscurazione 1% 50%), oscurazione intensa = 2 (percentuale di oscurazione 50% 100%)).

Tabella 3.1: Campi annotation dataset

Nome	Descrizione
`<object-class>`	La classe dell'oggetto, identificata da un numero intero che rappresenta l'indice della classe nell'elenco delle classi.
`<x>`	La coordinata x dell'angolo in alto a sinistra del bounding box normalizzata rispetto alla larghezza dell'immagine.
`<y>`	La coordinata y dell'angolo in alto a sinistra del bounding box normalizzata rispetto all'altezza dell'immagine.
`<width>`	La larghezza del bounding box normalizzata rispetto alla larghezza dell'immagine.
`<height>`	L'altezza del bounding box normalizzata rispetto all'altezza dell'immagine.

Tabella 3.2: Descrizione delle annotazioni per YOLO

È importante rimarcare che YOLO utilizza una convenzione differente per le misure che consentono di individuare i bounding box degli oggetti, poichè proprio per questa ragione è stato necessario eseguire un preprocessing specifico che modificasse le annotazioni, al fine di adattarle alle caratteristiche richieste da YOLO. Questo preprocessing è stato fondamentale per garantire che le coordinate dei bounding box fossero coerenti con le convenzioni adottate da YOLO, per permettere inizialmente una corretta identificazione degli oggetti durante l'addestramento e l'inferenza del modello poi.
Prima di procedere all'addestramento del modello, è stato necessario effettuare anche una normalizzazione dei parametri contenuti nelle annotazioni già adattate a YOLO. Questo aspetto sarà trattato in modo più dettagliato nel paragrafo successivo.

3.3 Preprocessing

Nei sottoparagrafi seguenti verranno illustrate le operazioni eseguite sui dati del VisDrone-dataset al fine di addestrare il modello YOLO. Queste operazioni sono state fondamentali per poter soddisfare i requisiti di addestramento del modello YOLO(sezione 3.2). Inoltre, saranno presentati i dettagli relativi alla suddivisione del dataset, apportata per consentire l'applicazione di vari scenari di apprendimento federato. Questa suddivisione del dataset ha permesso di creare scenari realistici, in cui diversi dispositivi contribuiscono all'addestramento del modello YOLO utilizzando i propri dati locali.

3.3.1 Modificche alle annotation

Al fine di adattare i dati al modello YOLO, sono state eseguite alcune operazioni di manipolazione su di essi, con particolare attenzione alle annotazioni. Nel dataset originale, le annotazioni erano strutturate in un formato diverso rispetto a quello richiesto da YOLO (tab 3.1 ,tab 3.2), rendendo necessaria la creazione di nuove annotazioni conformi alle specifiche di YOLO. Per affrontare questa sfida, è stato prima necessario comprendere nel dettaglio la struttura delle annotazioni nel database e successivamente applicare opportune operazioni matematiche per trasformare i valori delle coordinate delle bounding box e di altre informazioni nelle forme richieste da YOLO.

Per semplificare il processo di manipolazione delle annotazioni è stato sviluppato un algoritmo dedicato che opera eseguendo i seguenti step:

1. Individuazione della classe dell'oggetto: per ogni riga dell'annotazione (dove ogni riga rappresenta un oggetto contenuto nell'immagine), il campo nella posizione 5 della riga (Figura 3.6) rappresenta proprio la classe di appartenenza dell'oggetto. Una volta individuato, il valore della classe viene riportato nella posizione 0 della stessa riga in un nuovo file con lo stesso numero di righe.

2. Calcolo della coordinata x della bounding box dall'angolo in alto a sinistra: anzitutto, è necessario calcolare l'altezza e la larghezza dell'immagine a cui fa riferimento l'intera annotazione. L'altezza dell'immagine è indicata come H = img.shape(0) e la larghezza come W = img.shape(1). Successivamente, la coordinata x della bounding box viene calcolata come:

$$\text{x} = \frac{2 \times \text{bbox\_left} + \text{bbox\_width}}{2 \times W}$$

. Questa coordinata viene riscritta nella posizione 1 della stessa riga nel nuovo documento di testo.

3. Calcolo della coordinata y della bounding box dall'angolo in alto a sinistra: la coordinata y viene calcolata applicando l'operazione seguente:

$$\text{y} = \frac{2 \times \text{bbox\_top} + \text{bbox\_height}}{2 \times H}$$

Questa coordinata viene riportata nella posizione 2 della stessa riga nel nuovo documento di testo.

4. Calcolo della larghezza della bounding box: la larghezza della bounding box viene calcolata come:

$$\text{width} = \frac{\text{bbox\_width}}{W}$$

Questa valore viene scritto nella posizione 3 della stessa riga nel nuovo documento di testo.

5. Calcolo dell'altezza della bounding box: l'altezza della bounding box viene calcolata mediante il rapporto di seguito:

$$\text{height} = \frac{\text{bbox\_height}}{H}$$

Questo valore viene riportato nella posizione 4 della stessa riga nel nuovo documento di testo.

Per ottimizzare i tempi, l'algoritmo esegue anche la normalizzazione delle coordinate durante la generazione delle nuove annotazioni. In YOLO, le bounding box che delimitano gli oggetti di interesse vengono definite dalle coordinate delle loro quattro estremità come è possibile vedere nella tabella 3.2. Tuttavia, queste coordinate potrebbero variare in base alle dimensioni dell'immagine di input. Si ricorre per questa ragione alla normalizzazione delle coordinate che viene effettuata per rendere le coordinate relative e indipendenti dalle dimensioni dell'immagine. La normalizzazione applicata prevede la divisione di ciascuna coordinata per la larghezza e l'altezza dell'immagine. Le formule utilizzate per la normalizzazione delle coordinate sono le seguenti:

$$x_{\text{normalizzata}} = \frac{x}{\text{larghezza immagine}}$$

$$y_{\text{normalizzata}} = \frac{y}{\text{altezza immagine}}$$

$$Width_{\text{normalizzata}} = \frac{y}{\text{altezza immagine}}$$

$$Hight_{\text{normalizzata}} = \frac{y}{\text{altezza immagine}}$$

Questa operazione è fondamentale per l'addestramento di YOLO. Normalizzare i dati significa ridimensionare le coordinate e le dimensioni delle bounding box in un intervallo compreso tra 0 e 1. Ciò permette al modello un apprendimento più efficace oltre che a migliorare la stabilità dell'addestramento. In questo modo, con una singola esecuzione, l'algoritmo si occupa sia della trasformazione delle annotazioni che della normalizzazione dei dati. Di seguito viene riportata una riga di file di annotazione prima fig.3.6 e dopo fig.3.7 l'applicazione dell'algoritmo.

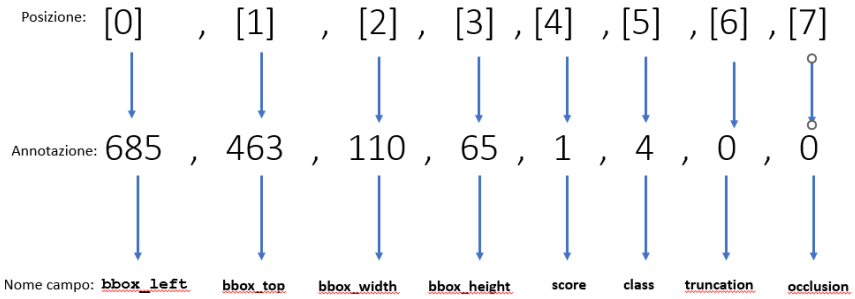

Figura 3.6: Riga file di annotazione prima delle modifiche.

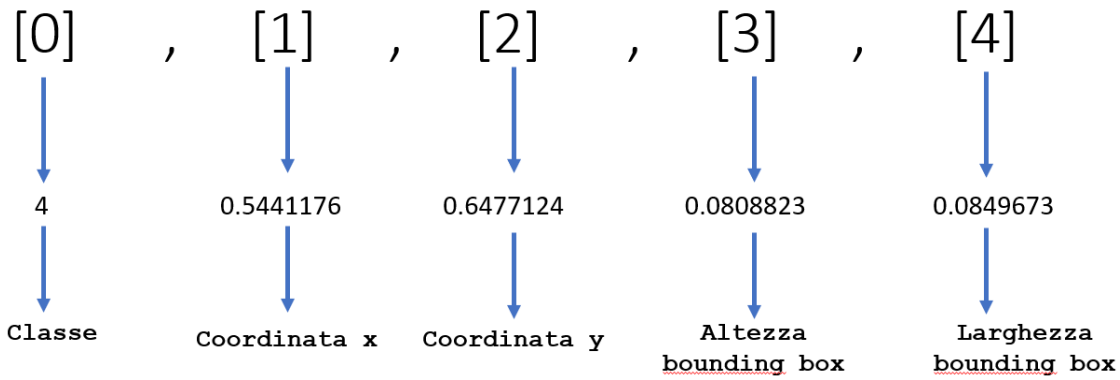

Figura 3.7: Riga file di annotazione dopo l'applicazione dell'algoritmo.

Quanto derivato dall'utilizzo dell'algoritmo mi ha consentito di ottenere delle annotation che permettono di identificare un'oggetto in un immagine, e quindi utilizzabili per l'addestramento del modello YOLO. Di seguito un immagine esemplificativa:

Figura 3.8: Annotation formato Yolo applicate

In fine l'ultima modifica che ho apportato alle annotation è stata l'eliminazione di due classi, la classe regioni ignorate (0) e la classe altri (11), in modo da rendere il processo di identificazione di quelle restanti più preciso. Ho cercato quindi di non generare ambiguità tra le diverse classi, cosa che ho notato in alcuni casi, per esempio nel momento in cui il modello identificava una persona (classe 1) a cavallo di una bicicletta (classe 3) ma la identificava come altri (classe 11, quindi rimossa).

3.3.2 Operazioni per utilizzo in federated learning

Per testare l'efficacia dell'addestramento federato, è stato necessario suddividere il dataset iniziale in dataset più piccoli, che sono stati poi distribuiti tra i diversi host. Dato lo squilibrio nella distribuzione delle classi di oggetti, ho implementato un codice che consente di distribuire i dati tra i vari host in base allo scenario che desidero rappresentare nei miei esperimenti.

Un esempio di come i dati possono essere distribuiti in un addestramento federato con tre host è illustrato nella figura 3.9 3.10 3.11. È importante sottolineare che i dati vengono completamente separati e l'accesso ai dataset più piccoli è fornito solo all'host a cui viene assegnato quel particolare dataset. Questo è in linea con il principio fondamentale sostenuto nell'addestramento federato, ovvero garantire la privacy a tutti gli utilizzatori.

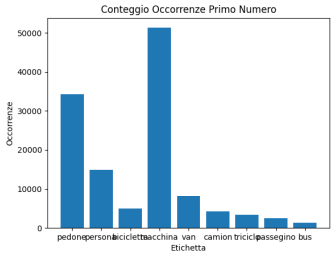

Figura 3.9: Distribuzione dataset host 1

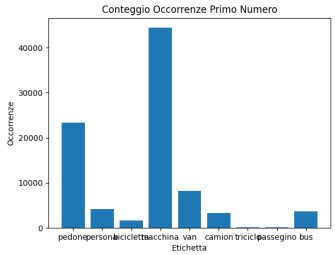

Figura 3.10: Distribuzione dataset host 2

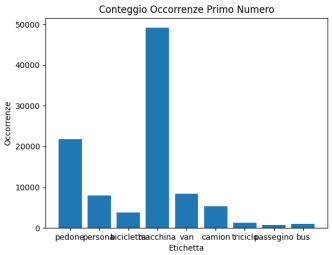

Figura 3.11: Distribuzione dataset host 3

In questo caso specifico ho scelto di mantenere una distribuzione simile a quella del dataset utilizzato nel training centralizzato. È abbastanza ovvio che si otterrà una maggiore precisione nel rilevamento degli oggetti appartenenti alle classi maggiormente popolate. Tuttavia, questo risultato è atteso in qualsiasi addestramento in cui non viene inclusa una discriminante specifica per la classe maggiormente popolate, poiché queste classi sono popolate in maniera significativamente maggiore rispetto alle altre, portando il modello ad enfatizzarla naturalmente.

Capitolo 4

Implementazioni SmartFed

In questo capitolo, illustrerò l'approccio che ho adottato per implementare le architetture di CFL (Centralized Federated Learning) (immagine 1.1) e DFL (Decentralized Federated Learning)(immagine 1.1). Per effettuare queste implementazioni non mi sono servito di alcun framework preesistente sviluppato specificamente per il federated learning. Il mio obiettivo è, infatti, quello di affrontare direttamente le logiche di funzionamento del federated learning, cercando di gestirle in modo autonomo e personalizzato, al fine di fornire un nuovo Framewrk facilmente utilizzabile per l'addestramento di qualsiasi tipo di modello, sfruttando un contesto federato, chiamerò questo nuovo framework SmartFed.
Per poter realizzare ciò, ho gestito manualmente tutte le comunicazioni tra i partecipanti, costruendo endpoint ed utilizzando API REST. Questo approccio mi ha permesso di creare uno scenario realistico, sebbene gli esperimenti siano stati eseguiti tutti sullo stesso computer. Ho distribuito quindi i vari nodi partecipanti all'apprendimento sul mio PC, consentendo loro di accedere solo alla porzione del dataset che gli era stata assegnata.
Per far fronte anche alle limitazioni hardware associate, ho sviluppato un processo di apprendimento federato asincrono per entrambe le architetture. Ciò significa che i diversi nodi possono procedere con l'addestramento singolarmente e quindi accedere alle risorse necessarie per portarlo al termine uno alla volta, evitando sovraccarichi. Tuttavia, benché questa scelta abbia comportato un notevole aumento dei tempi di addestramento, al contempo mi ha ad ogni modo permesso di condurre dei test realistici con risultati significativi.

4.1 FL centralizzato implementazione

Come spiegato nel paragrafo 1.2.1 questa architettura si compone di due entità fondamentali:

- i **client** che eseguono l'addestramento del modello in locale;
- il **server** che ha il ruolo fondamentale di gestire le comunicazioni tra i diversi partecipanti e di aggregare i modelli in modo da crearne uno un globale.

4.1.1 Client

Inoltriamoci ora nel dettaglio di come ho implementato lo script (sfruttando il linguaggio Python) che è stato poi eseguito da ogni client per poter partecipare all'addestramento federato.
Di seguito un diagramma illustrativo dei vari step eseguiti dallo script deployato su ogni client Figura:4.1:

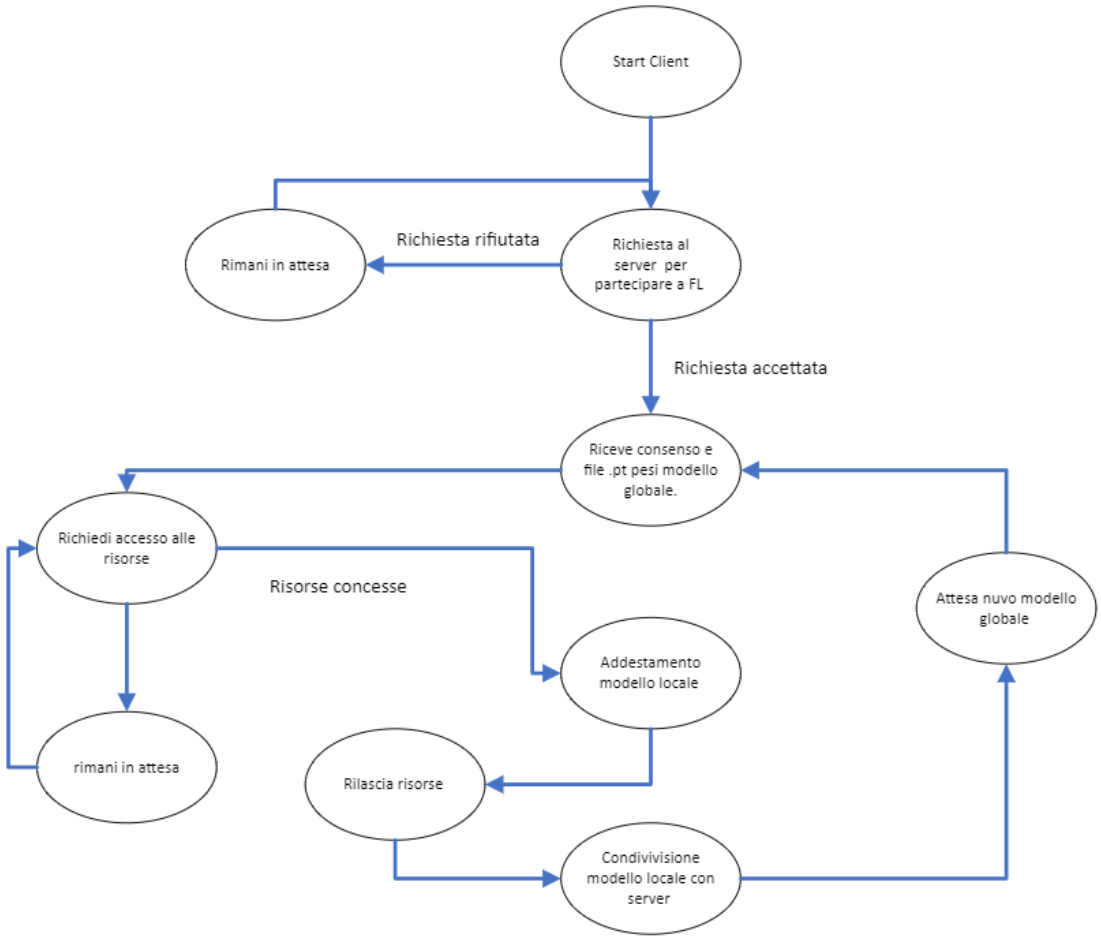

Figura 4.1: Stati client

Questo diagramma chiarisce le fasi e le transizioni che il client attraversa durante l'addestramento federato centralizzato. Ogni stato rappresenta una specifica attività o condizione che può essere assunta durante il processo di addestramento. L'idea è che ogni client intenzionato a partecipare dovrà collegarsi ad uno share point da cui potrà scaricare il codice ed infine consentire al codice di accedere al proprio database salvato in locale. Dopo aver concesso l'accesso ai propri dati il codice eseguirà le seguenti operazioni:

1. **Inizializzazione**: il client esegue lo script creando un oggetto chiamato Client che verrà inizializzato con i seguenti attributi:

   - **ID:** questo parametro che si presenta inizialmente come vuoto (stringa vuota), in seguito alla richiesta al server di partecipare alla federazione servirà ad identificare in maniera univoca il partecipante mediante un numero;
   - **ServerIP:** tale parametro permette di tracciare l'ip del server;
   - **ServerPort:** è il parametro contenente la porta del server a cui è possibile eseguire delle chiamate;
   - **GlobalWeight:** questo parametro inizializzato come `None`, in seguito conterrà sempre i riferimenti all'ultimo modello aggregato dal server. Verrà popolato solo dopo il termine del primo round;
   - **NROUND:** questo valore indica il numero massimo di round ai quali il client è disposto a partecipare;
   - **round:** il parametro rappresenta il numero di addestramenti che ha eseguito il client fino a questo momento. Anche questo valore è inizializzato a 0;
   - **WorkerWeight:** variabile d'oggetto contenente i riferimenti al file .pt, in cui sono riportati i pesi dell'ultimo addestramento eseguito dal Client. Il parametro viene inizializzato con una stringa vuota;
   - **Sample:** questo parametro serve per tenere traccia del numero di campioni posseduto dal client. Viene inizializzato chiamando la funzione `CountSample()`, che si occupa di contare il numero di oggetti presenti nelle immagini disponibili per l'addestramento locale;
   - **INFED:** è un parametro di stato, in quale specifica se il client sta già partecipando all'addestramento federato. Inizializzato a **False**, diventerà `True` solo dopo che il server avrà inserito il client nella federazione.

   In seguito all'inizializzazione dell'oggetto `Client`, il partecipante è pronto per procedere con la richiesta di ammissione al sistema federato.

2. **Ammissione alla federazione**: a questo punto viene richiamata la funzione `AddFedd` attraverso la quale si esegue una richiesta `GET` al server per l'annessione del client al sistema federato. Questa funzione invia una request HTTP al server per richiedere l'inserimento del client all'addestramento federato. La richiesta GET passa come variabile della uri il numero di campioni a disposizione del Client. Al termine di questi step potrebbero presentarsi tre plausibili situazioni:

   (a) **Caso ottimale:** il server risponde correttamente fornendo al client l'ID univoco che lo identifica all'interno del sistema federato. L'ID viene memorizzato nell'attributo `Client.ID`.

   (b) **Rifiuto:** nel caso di un rifiuto da parte del server, il client attende 5 minuti per poi rieseguire la richiesta fino a quando non verrà accettato.

   (c) **Indisponibilità del server:** nel caso in cui, invece, il server non dovesse essere disponibile, il client attende 5 minuti prima di riprovare ad eseguire un ulteriore richiesta. Reitererà questa operazione fino a quando il server non sarà nuovamente disponibile

3. **Pronto per eseguire l'addestramento**: in questa fase viene richiamata la funzione `Ready` al fine di informare il server che il client è pronto ad eseguire un nuovo addestramento. Questa funzione serve a sincronizzare i client partecipanti affinché inizino l'addestramento partendo dallo stesso modello globale, ossia quello ottenuto dall'ultimo round eseguito dal server. Ovviamente nel caso in cui un nodo si dovesse aggiungere in seguito all'inizio dell'addestramento, questo partirà direttamente dall'ultimo round al quale il sistema è arrivato. In più, questa funziona, si occupa di verificare che ogni client riceva il modello globale una sola volta per ogni round, e che quindi possa comunicare al server un solo modello locale. Nel caso in cui il client dovesse aver già ricevuto l'ultima versione del modello globale, sarà il server a comunicarglielo ed il client dovrà restare in attesa fino alla creazione di un nuovo upgrade del modello.

4. **Richiesta del modello Globale**: dopo aver verificato di non aver ancora ricevuto il modello globale calcolato nell'ultimo round, il Client lo richiede attraverso la funzione `GetPTfile`. Il Server fornisce quindi il file in questione che verrà salvato nell'attributo `Client.GlobalWeight`.

5. **Richiedi risorse**. Prima di procedere con l'addestramento, viene richiamata anche la funzione `RichiediGPU`. Questa funzione invia una richiesta al server tramite una request di tipo **GET**, passando come variabile nell'url l'ID del client. L'obbiettivo è quello di ottenere un'allocazione temporanea della GPU al fine di poter procedere

con l'addestramento. Il server inoltre gestisce l'allocazione delle risorse tra i client partecipanti per garantire una distribuzione equa delle GPU. Il sever potrebbe anche concedere l'utilizzo della GPU al client se la coda di richieste di accesso alla GPU è vuota, o se il prossimo ad aver diritto di accesso alla risorsa è il client che esegue la richiesta. In questi due casi la risposta del server sarà un'http status code 200 e quindi il client potrà procedere con l'addestramento,
Nel caso in cui al server non sia ancora possibile assegnare la GPU al client in questione la risposta sarà un'HTTP status code uguale a 201. Questo vorrà dire che la GPU è momentaneamente impegnata, quindi il client dovrà mettersi in attesa e rieseguire la richiesta dopo 5 minuti fino a quando non riuscirà ad accedere alla risorsa desiderata.
La decisione di inserire questa funzionalità nel server verrà spiegata in seguito.

6. **Addestramento del modello**: in questo step il client addestra il proprio modello utilizzando i pesi globali ottenuti. L'addestramento viene eseguito dalla funzione `Train`, la quale utilizza il modulo `WorkerYolo` per addestrare il modello. In quest'ultima funzione vengono inoltre specificati anche i parametri di addestramento come la dimensione del batch ed il numero di epoche.

7. **Salvataggio dei nuovi pesi**: dopo l'addestramento, i nuovi pesi sono salvati in locale in una cartella corrispondente al round di addestramento corrente, in modo tale da poter tenere traccia dei round ai quali ha partecipato.

8. **Rilascio della GPU**: al termine dell'addestramento, viene richiamata la funzione `LiberaGPU` che tramite una richiesta `POST` in cui viene passato l'id del client, informa il server della possibilità di assegnare la risorsa ad un nuovo richiedente.

9. **Invio dei pesi al server**: il client mediante l'utilizzo della funzione `SendPtFile` invia i nuovi pesi ottenuti al server, i quali vengono poi inclusi nella richiesta HTTP come file binario. Il server una volta ricevuti i pesi dal client li utilizza per calcolare i nuovi pesi globali per il round successivo.

10. **Avanzamento dei round**: il client incrementa il contatore dei raund e ripete il processo dal punto 3 fino al momento in cui non gli verrà messo a disposizione un nuovo modello globale.

La gestione della GPU (di cui ai punti 5 ed 8) è stata necessaria in quanto tutte le entità che partecipavano alla federazione sono state eseguite sullo stesso PC. Ovviamente questa è una situazione che non si verificherebbe mai in un caso reale di addestramento federato, ma per svolgere dei test sulle metodologie implemetate ho dovuto optare per

questa soluzione . Inoltre la gestione della GPU, nel mio caso, è stata fondamentale visto e considerato che le risorse del mio PC, una sola GPU RTX 3060, non sarebbero state sufficienti per consentire l'esecuzione simultanea dell'addestramento su diversi host.
Questa configurazione non influisce sui risultati dei miei test poiché, anche se i partecipanti all'apprendimento federato condividono le stesse risorse, i principi fondamentali del FL come la non condivisione dei dati, vengono comunque garantiti. Di fatto quando si parla di risorse, si fa riferimento solo alla GPU, e non ai dataset (quindi alla parte di memoria) utilizzati per l'addestramento dei modelli distribuiti sui diversi nodi.

4.1.2 Server

In un addestramento federato centralizzato, la componente centrale che governa il funzionamento dell'addestramento è il server, il quale svolge un ruolo fondamentale nella coordinazione e nell'aggregazione delle informazioni ottenute dai nodi partecipanti. Per tale ragione ci si può aspettare che le logiche seguite dal server siano più complesse rispetto a quelle viste in precedenza. Nell'immagine 4.2, riportata di seguito, viene presentato un diagramma che raffigura le varie operazioni eseguite dal server:

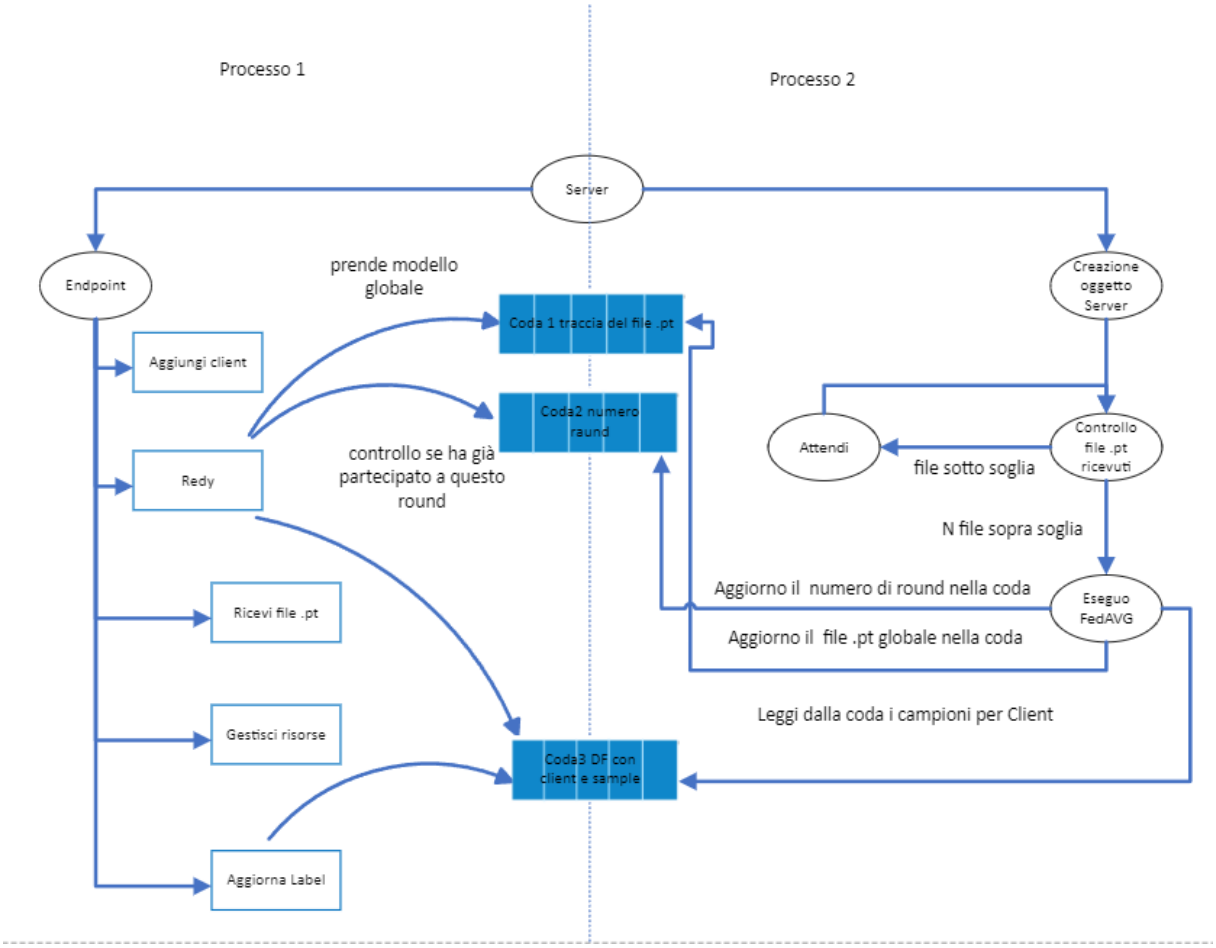

Figura 4.2: Stati server

Questo diagramma illustra le fasi e le transizioni che il server attraversa durante l'addestramento federato centralizzato. Ogni stato rappresenta una specifica attività o condizione che il server può assumere durante questo processo.
Nell'implementazione del server, sono stati utilizzati due processi eseguiti in modo asincrono. Mentre il primo è responsabile di metter sempre a disposizione dei partecipanti all'addestramento gli endpoint necessari per comunicare con il server, il secondo processo, eseguito in background, si occupa di appurare il numero di client partecipanti all'adde-

stramento oltre che di esegue l'aggregazione dei modelli per crearne uno globale. Questo ultimo passaggio avviene solo nel momento in cui un numero sufficiente di partecipanti ha messo a disposizione il risultato del loro addestramento.
Per consentire la comunicazione tra i due processi e quindi il passaggio di informazioni tra questi, sono state implementate due code. Al momento dell'esecuzione del codice che si occupa della creazione del server viene immediatamente istanziato l'oggetto `Server`, che contiene i seguenti attributi:

- **ip**: è l'attributo in cui viene registrato l'ip esposto ai Client.
- **GlobalWeight:** questo attributo memorizza l'ultimo percorso in cui è stato salvato l'ultimo modello globale creato e viene inizializzato con il riferimento al modello preaddestrato di YOLO. Pertanto, all'inizio del processo di addestramento federato, tutti i client che parteciperanno dal round 0, eseguiranno il primo ciclo di addestramento utilizzando i pesi del modello YOLO addestrato sul dataset COCO. Questo approccio consente di avviare l'addestramento federato da uno stato iniziale comune, fornendo una base solida per il successivo miglioramento collaborativo del modello.
- **NumWorker**: questa variabile d'oggetto tiene traccia del numero di partecipanti all'addestramento federato.
- **round:** è una proprietà dell'oggetto Server si occupa di tener traccia del round corrente. Viene inizializzata a 0.
- **accessoGPU:** si tratta di un attributo contenente una lista inizializzata come vuota. Il server sfrutterà poi questa lista per gestire le richieste di accesso alla GPU da parte dei vari partecipanti.
- **dataframe:** in questa variabile d'oggetto viene salvato un dataframe contenente l'id dei partecipanti ed a ciascuno di questi è poi associato un numero di campioni. Questo attributo è utile per tener traccia del numero di campioni per ciascun client, al fine di applicare l'algoritmo FedAVG in maniera appropriata.
- **WorkerCheck:** si tratta di un attributo contenente una lista vuota, infatti, viene svuotata ad ogni round. É utilizzata per tener traccia di quale client ha già ricevuto il modello globale, in modo tale da non creare ridondanza inviando due volte lo stesso modello. Inoltre con questa variabile il server è in grado di comprendere quale client sta partecipando al round in corso ed evitare così eventuali ritrasmissioni dello stesso modello locale dal client.

Dopo l'inizializzazione dell'oggetto Server, per creare un server in grado di gestire le richieste dei vari client ed allo stesso tempo aggregare i modelli, è stato necessario l'utilizzo

della libreria `Multiprocessing` di Python. Questa libreria consente di gestire in maniera semplice due processi eseguiti in maniera asincrona. Il primo processo serve ad esporre gli endpoint per permettere a tutti i client di comunicare con il server, mentre il secondo si occupa della gestione e dell aggiornamento dell'oggetto `server`. I due processi riescono a cumunicare tra di loro tramite delle code.
Nello specifico i due processi eseguiranno le operazioni enunciate di seguito:

PROCESSO 1
Questo processo, come suddetto in breve, si occupa della creazione del server web, creato attraverso il framework Flask. Il server espone diversi endpoint ai quali i client possono fare delle richieste http per scambiare informazioni con quest'ultimo. Gli endpoint che ho implementato, nello specifico, sono:

- /**inizio**, nel momento in cui si effettua una richiesta get a questo endpoint viene richiamata la funzione `Inizio()`. Questa funzione si occupa di:

  1. Incrementare il valore della variabile `Server.NumWorker` di 1 all'interno dell'oggetto Server, per tracciare il numero di client disponibili nel sistema federato.
  2. Estrarre il valore del parametro "sample" presente nell'url della richiesta GET utilizzando request.args.get('sample'). Nel caso in cui questo parametro non dovesse essere disponibile la funzione ritorna con un errore 402.
  3. Assegnare alla variabile id il numero attuale di client presenti nel sistema fino a quel momento. Questo permette di generare un identificatore univoco per il client che ha eseguito la richiesta.
  4. Aggiungere una nuova riga al dataframe contenuto nell'oggetto Server, la quale tiene traccia del numero di Sample per partecipante all'addestramento, creando la coppia ID Sample.
  5. Inserire nella coda Sample il dataframe aggiornato, in modo tale che il processo due (spiegato in seguito) abbia sempre a disposizione l'ultima versione del dataframe.

- /**redy**, in questo caso, invece, la funzione:

  1. Controlla se nelle code sono stati inseriti degli aggiornamenti.
     Verifica quindi se è stato calcolato un nuovo modello globale, ed in caso di esito positivo inizia a fornire, ai client che lo richiedono, il nuovo modello. Verifica inoltre il round al quale si trova il processo 2 in modo da comunicarlo ai vari client.

2. Raccoglie l'elenco di tutti i client che vogliono partecipare al round correte. Questo controllo permette di evitare casi in cui lo stesso client invia al server due volte il risultato dell'addestramento effettuato in locale.
3. Questo endpoint può restituire una delle seguenti risposte:
   (a) **200:** sta ad indicare che il client può partecipare a questo round, in quanto non ha ancora inviato i dati per il ciclo in corso.
   (b) **201:** denota il fatto che il client ha già partecipato al round in corso, deve perciò attendere che venga calcolato un nuovo modello globale.

   Questi controlli sono necessari in quanto, trovandoci in una situazione in cui tutti i worker addestrano in maniera asincrona i propri modelli, il primo che terminerà l'addestramento richiederà immediatamente di partecipare al round successivo che però non sarà ancora pronto in quanto il server sarà ancora in attesa dei dati degli altri client.

- /**ricevePtFile**, questo tipo di funzione:

  1. Controlla la coda in cui il processo 2 ha salvato il numero del round in cui si trova.
  2. Riceve i file pt dai diversi client e li salva nella cartella del round corrente, in modo che il processo due possa utilizzare tutti i file presenti in quella cartella per calcolare il nuovo modello Globale.
  3. Restituisce come esito 200 se l'operazione è andata a buon fine.

- /**sendGlobal**
  Questo endpoint si occupa di fornire l'ultimo modello calcolato ogni qualvolta un client richiede il modello Globale.

- /**GPU**
  Si tratta di un endpoint che può ricevere due tipi di richieste:

  1. **GET:** si verifica questo tipo di richiesta nel momento in cui un client necessita l'accesso alla GPU. Vengono quindi eseguiti dei controlli sulla lista che tiene traccia delle richieste di accesso alla GPU.
     – Se il client è già presente nella coda ma non è ancora il suo turno per prendere il controllo della GPU, l'endpoint restituisce in risposta il codice 201 (Created).
     – Se il client che esegue la richiesta non è ancora nella lista, viene aggiunto e l'endpoint restituisce 201 se la risorsa è già assegnata ad un altro partecipante.

- Se il client che esegue la richiesta è già presente nella lista e si trova in posizione 0 , allora è il momento di concedergli l'utilizzo della GPU, e in questo caso l'endpoint restituisce una risposta con codice 200 (OK).

2. **POST:** una volta terminato di utilizzare la GPU, i client eseguiranno una richiesta POST a questo endpoint ed in questo modo verranno eliminati dalla coda di accesso. Il server potrà quindi procedere ad assegnare l'utilizzo della risorsa al prossimo partecipante nella lista.

- **/Aggiorna**:
  L'endpoint in questione gestisce gli aggiornamenti dei dati inviati dai diversi client. Ad esempio, se uno dei client dispone di nuovi campioni da utilizzare per l'addestramento, può servirsi di questo endpoint per inviare il nuovo numero di label. Prima di inviare il modello addestrato con questi nuovi dati, il client deve comunicare la loro numerosità al fine di tener conto di questa variazione durante il calcolo del modello aggregato. I nuovi dati vengono presi in considerazione dal server nel processo di calcolo del modello globale per incorporare le informazioni aggiornate fornite dal client.

PROCESSO 2

Questo secondo processo si occupa dell'esecuzione della funzione `Federato()` dell'oggetto Server. La funzione è stata creata per gestire tutte le operazioni che devono essere svolte nel server, ovvero:

- **Avvio processo**
  Quando viene eseguito l'avvio del processo, gli viene passata la funzione `Server.Federato()`, che viene eseguita in background. La funzione `Server.Federato()` rappresenta la logica principale del server per l'addestramento federato centralizzato.

- **Metodo Federato**
  All'interno del metodo, è presente un ciclo che verrà ripetuto per il numero di round che si desidera eseguire. Questo ciclo effettua controlli periodici per verificare se sono stati ricevuti i file necessari ad avviare l'operazione di aggregazione dei modelli. In particolare, il server verifica se sono stati ricevuti un numero sufficiente di file contenenti i pesi dei modelli addestrati in locale dai singoli client. Solo quando è stato raggiunto un numero adeguato di file è possibile avviare l'algoritmo di aggregazione, per combinare e consolidare i pesi dei modelli locali.

  - **File disponibili:**
    Se i file sono disponibili, il server creerà una nuova directory in cui salverà il

risultato dell'aggregazione. Il nome della cartella ci permetterà di tener traccia del raund,in quanto il nome sarà costituito da "raund" più il numero effettivo del ciclo. In seguito viene richiamata anche la funzione `AVGfed` che si occupa dell'aggregazione dei diversi modelli e quindi del calco dei nuovi pesi globali. I risultati della funzione `AVGfed`, ovvero i nuovi pesi, vengono quindi inviati ai due endpoint che verranno richiamati dai client partecipanti all'addestramento tramite le code `g` e `q`.

La funzione `AVGfed` richiede che gli vengano passati 3 parametri: una lista contenente i path dei file in cui sono riportati i pesi dei modelli locali, il numero del Round corrente, il numero di sample per ogni client che sta partecipando all'addestramento di quel round. Prima di richiamare questa funzione, è necessario verificare l'aggiornamento riguardante il numero di sample.

– **File non disponibili:**

Il processo in questo caso rimane in attesa e ogni N minuti verifica se si è ricevuto un numero di file sufficienti per avviare il processo di aggregazione.

Al termine del ciclo viene eseguito un test sui diversi modelli calcolati ad ogni round per eleggere quello con l'accuratezza maggiore.

4.1.3 Queues (code)

Ogni queue è una struttura dati che consente la comunicazione tra i processi, fornendo un meccanismo FIFO (First-In-First-Out) Figura:4.3. FIFO permette a tutti i processi di inserire oggetti nella coda per poterli poi prelevarli successivamente.

Figura 4.3: Coda FIFO

La classe Queue del modulo multiprocessing offre una funzionalità di gestione sicura delle code, infatti, garantisce la sincronizzazione e la condivisione dei dati tra i processi in modo thread-safe. Ciò significa che può essere utilizzata in un ambiente concorrente, in cui più processi possono accedere e manipolare la coda contemporaneamente, senza causare problemi di sincronizzazione. Come esposto nel paragrafo precedente per lo sviluppo del server è stata utilizzata questa struttura, con l'obiettivo di garantire la comunicazione dei due processi in esecuzione in modo asincrono sul server.

4.2 FL decentralizzato implementazione

L'implementazione dell'architettura di DFL è risultata essere molto più complessa rispetto a quella trattata all'inizio del presente capitolo, infatti, nel DFL ogni singolo nodo deve eseguire tutte le operazioni che costituiscono l'architettura in questione (cfr paragrafo 4.1.2).
Lo scenario che ho voluto modellare prevede che un client decida di partecipare ad un processo di addestramento federato connettendosi ad uno SharePoint, da cui potrà scaricare il codice che gli permetterà di partecipare alla federazione.
I file sopraindicati, caricati sullo sharepoint, nello specifico sono:

1. **File IP-DFL.txt :** contenete gli indirizzi di alcuni nodi che partecipano già all'apprendimento federato;

2. **Modello:** l'algoritmo di machine learning che si sta addestrando utilizzando il processo di addestramento federato;

3. **Script Worker.py per partecipare alla federazione:** script per mezzo del quale il client potrà far richiesta di partecipare all'apprendimento federato. Logicamente per far ciò dovrà esporre un proprio indirizzo IP in modo da poter essere contattato dagli altri client.

   Nei sottoparagrafi seguenti illustrerò in che modo ho deciso di implementare questa architettura.

4.2.1 Client e Server

Assumiamo che esista una federazione, e che questa stia già addestrando un modello YOLO. Un nuovo host che desidera partecipare all'addestramento deve collegarsi ad uno SharePoint, da cui potrà scaricare tutti i dati necessari per parteciparvi.
In seguito al download, il client dovrà eseguire tre operazioni:

1. **Esposizione dell'indirizzo IP:** il client espone un indirizzo IP per poter essere raggiunto dagli altri partecipanti. Quindi inserisce questo indirizzo nel campo IP della classe Client presente nel file `Worker.py` scaricato dallo SharePoint.

2. **Accesso al dataset:** il client permette al file `Worker.py` di accedere al proprio dataset memorizzato localmente.

3. **Esecuzione di Worker.py:** il client esegue il codice, che invierà una richiesta a uno degli altri client già partecipanti alla federazione presente nella lista `IP-DFL.txt`.

In seguito alla risposta, il nuovo client farà parte del processo di apprendimento federato.

Il funzionamento di ogni client è simile a quello del server presentato nel sottoparagrafo 4.1.2 , con alcune accezioni. In questo caso, infatti, ogni client si occupa dell'aggregazione dei diversi modelli locali (condivisi dai partecipanti) e dell'esecuzione dell'addestramento del proprio modello locale con i dati a propria disposizione. Solo al termine degli N round si valuterà il modello globale con la migliore accuratezza.
Entrando nel dettaglio di quello che effettivamente avviene con l'esecuzione dello script `Worker.py`, nell'immagine 4.5 è presente una raffigurazione grafica di tutte le operazioni ed i controlli che questo compie. In linea con l'implementazione del Server presentata nel paragrafo 4.2, anche in questo caso è stato necessario l'utilizzo di due processi eseguiti in parallelo. Saranno ora descritte tutte le operazioni eseguite dai due processi:

PROCESSO 1
Il processo uno, come nel caso del server presentato in precedenza, si occupa di esporre gli endpoint con l'obiettivo di permettere ai diversi partecipanti di comunicare tra loro. I vari endpoint che ho provveduto ad implementare sono i seguenti:

- **/AggiungiDFL**
  Questo endpoint gestisce le richieste provenienti dai nuovi host che desiderano partecipare al processo di addestramento federato. Per contattare questo endpoint, il nuovo host deve effettuare una chiamata POST, fornendo il proprio indirizzo IP, la porta che desidera utilizzare per le comunicazioni ed il numero di campioni presenti nel suo dataset locale.
  Se le informazioni sono state fornite correttamente, il metodo invocato dall'endpoint esegue una serie di controlli, tra cui verificare l'assenza di un nodo con lo stesso indirizzo IP. Successivamente, trasmette a tutti gli altri client le informazioni del nuovo host, al fine di stabilire una connessione peer-to-peer tra i partecipanti al processo di addestramento federato. Infine, aggiorna il file locale `IP-DFL.txt` con l'aggiunta del nuovo host e restituisce un codice di stato HTTP 200. L'immagine di seguito raffigura le operazioni svolte da questo endpoint fig. 4.4

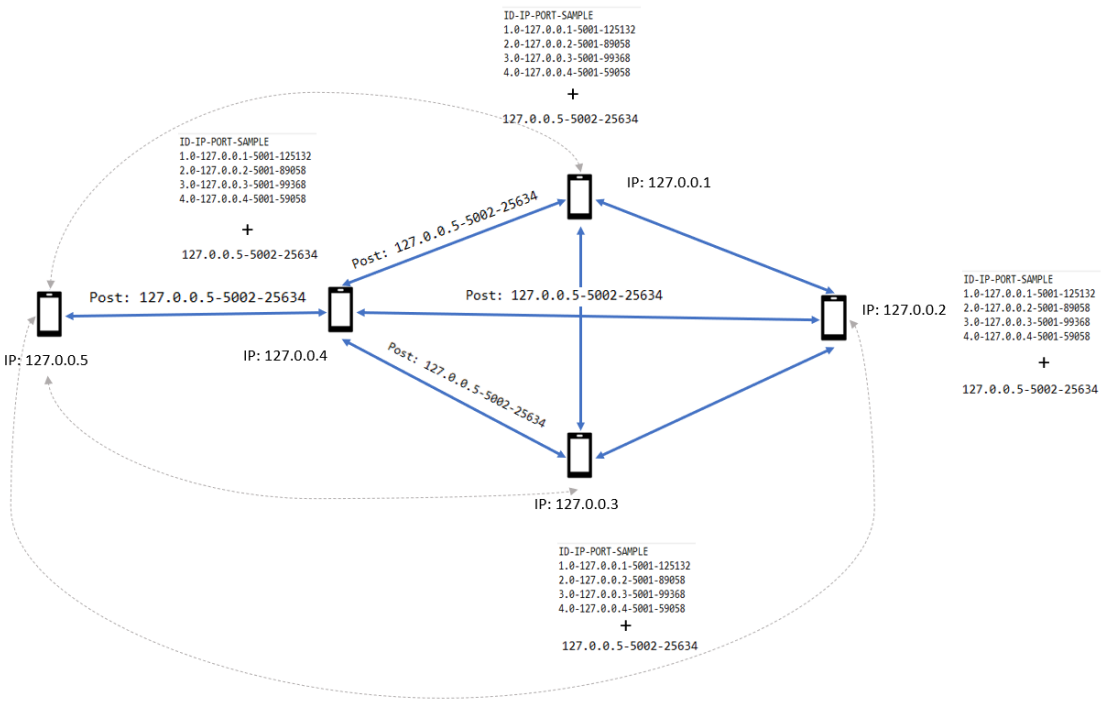

Figura 4.4: Aggiunta al sistema federato decentralizzato

- **/RiceviPT:**
  Questo endpoint riceve il file contenente i pesi dei modelli addestrati, o aggregati, dai nodi partecipanti all'addestramento federato. Quando viene chiamato, l'endpoint richiama la funzione `RiceviPT()`, la quale esegue un controllo sul tipo di file contenuto nel messaggio, verificando che abbia l'estensione ".pt". In un secondo momento, controlla l'header della chiamata, che deve includere tre campi:

  1. **IP del nodo chiamante**;
  2. **Tipo di file pt:** questo campo serve per determinare se il file condiviso rappresenta il risultato di un addestramento locale (Campo="train") di un nodo o l'aggregazione di diversi modelli eseguita sul nodo chiamante (Campo="agg");
  3. **Round:** il quale indica il round in cui è stato addestrato il modello che ha prodotto quei pesi o a quale round appartiene il modello aggregato.

  Dopo aver effettuato questi controlli, la funzione salva il file con il nome corrispondente all'indirizzo dell'host che lo ha inviato, per poi inserirlo nella cartella appositamente creata per il round specifico.
  In questo modo, si crea una struttura organizzata in cui i file dei pesi sono memorizzati separatamente per ogni round oltre che identificati in base all'indirizzo dell'host mittente. Questo approccio consente una gestione efficiente dei risultati

dell'addestramento locale dei nodi partecipanti all'addestramento federato, facilitando la successiva creazione del modello aggregato o l'aggiornamento del modello globale.
Nel caso in cui il file ricevuto sia il risultato dell'addestramento di uno dei nodi, verrà utilizzato per creare il modello aggregato. In caso contrario, sarà identificato dall'header `TypePT = "agg"`, e rappresenterà il modello aggregato con la precisione più alta sul nodo che ha eseguito la chiamata. In questo caso, questo modello verrà salvato nella cartella `BestAgg`, assegnandogli l'ip del nodo in modo da rendere sempre possibile l'identificazione.

- **/AggiornaIP-DFL:**
  L'endpoint richiamato dalla funzione `AggiungiDFL` permette di aggiornare la lista di indirizzi ip dei vari host partecipanti all'addestramento federato sul singolo host.

PROCESSO 2

Il processo due sempre in linea con le operazioni che vengono svolte dal server nel CFL, si occupa dell'aggregazione dei modelli. In questo caso però oltre all'aggregazione e quindi alla creazione del modello globale, si occupa anche dell'addestramento e della creazione del modello locale.
L'esecuzione del processo comporta i seguenti passaggi:

1. **Controllo della partecipazione:** appena avviato il processo esegue un controllo sulla variabile `Accettato`, ovvero una variabile dell'oggetto client che se settata a True indica che il client è già stato aggiunto al sistema federato. Se la variabile, invece, è uguale a False all'ora il processo richiama la funzione `Client.AggiungiDFL()` la quale apre il file `IP-DFL.txt`, precedentemente scaricato dall'endpoint e contenente gli indirizzi ip dei nodi già partecipanti al sistema federato. Esegue quindi una richiesta post all'endpoint /AggiungiDFL fornendo come body un file json contenente 3 parametri:

   (a) l'indirizzo IP del client;

   (b) il numero di campioni a disposizione per l'addestramento;

   (c) la porta sulla quale preferisce essere contattato.

   Se la chiamata restituisce un'http status 200 il client sarà aggiunto al sistema Federato. In più vedrà, tramite l'endpoint `/AggiornaIP-DFL` (precedentemente attivato), l'ip di tutti gli host che stanno partecipando all'addestramento federato aggiungersi al file `IP-DFL.txt`.

2. **Addestamento:** il client a questo punto esegue il primo addestramento del modello con i dati salvati in locale. Al termine del processo di addestramento salva il file .pt contenente i pesi del modello e lo condivide, attraverso il metodo `SendWeight("train")`, con tutti i nodi che partecipano all'addestramento.

3. **Attesa superamento soglia:** arrivati a questo punto il processo rimane in attesa, fino a quando non riceve un numero sufficiente di file contenenti i risultati dell'addestramento degli altri nodi tale da superare una soglia prefissata. Una volta raggiunta questa soglia, il processo esegue l'aggregazione dei modelli e crea il modello globale-locale. Questo modello sarà utilizzato per ripetere il processo di addestramento a partire dal punto 2.
   L'obiettivo della soglia prefissata è quello di assicurarsi che ci siano abbastanza risultati di addestramento da diversi nodi per effettuare un'aggregazione significativa e ottenere un modello globale rappresentativo delle conoscenze di tutti i partecipanti. Nel nostro caso abbiamo deciso di mantenere questa soglia statica, ciò non toglie che sia possibile apportare delle modifiche al codice per renderla dinamica, ad esempio facendo sì che si adatti al numero totale di Worker. In questo punto, inoltre, sarebbe possibile inserire un'ulteriore "logica" per tentare di individuare i file contenenti i risultati migliori è più sicuri.
   Una volta completata l'aggregazione dei modelli e creata la versione aggiornata del modello globale-locale, si procederà a ripetere il processo di addestramento ripartendo dal punto 2 ed utilizzando il nuovo modello come base.

4. **Valutazione:** una volta terminato il prefissato numero di round, varrà eseguita una valutazione sul dataset di test di ogni modello Globale-locale creato. Il modello che otterrà il livello di accuratezza maggiore sarà inviato a tutti gli altri partecipanti tramite il metodo `SendWeight("agg")`.

5. **Confronto dei risultati:** in questa ultima fase viene effettuato un confornto tra tutti i modelli che presentano la migliore accuratezza calcolata per ogni client. Al termine del confronto verrà identificato il modello che presenta un livello di accuratezza maggiore.

Nell'immagine fig:4.5 sono rappresentati i diversi stati che un client può assumere durante questo processo di addestramento. È importante sottolineare che nel caso appena descritto i dati generati dai due processi vengono salvati in locale sul client. Questo approccio di salvataggio immediato dei dati sulla memoria dell'host evita la necessità di utilizzare code o meccanismi di comunicazione complessi tra i processi. I dati sono salvati in un'area accessibile sia dal processo di addestramento locale e di aggregazione

dei modelli, che da quello che si occupa dell'esposizione degli endpoint, consentendo loro di accedere sempre alla versione più recente dei file senza problemi di sincronizzazione.

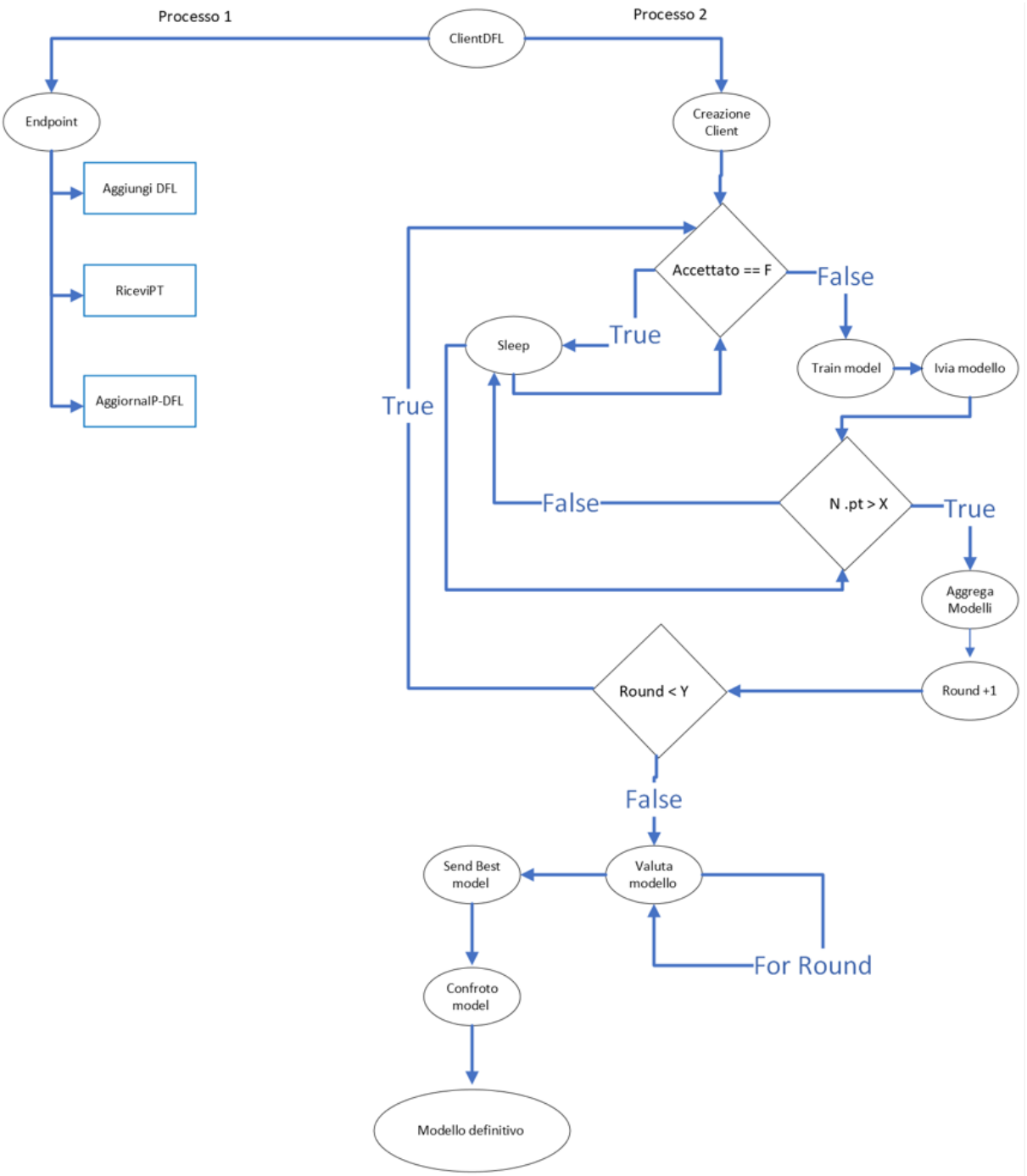

Figura 4.5: Algoritmo DFL

4.3 FedAVG

Nelle architetture presentate nei paragrafi precedenti di questo capitolo, viene utilizzata la funzione AVGForYOLO, importata dal modulo FedAVG. In entrambi i casi, ovvero CFL e DFL, viene richiamata questa funzione per eseguire il calcolo del modello globale (aggregazione modelli locali), che rappresenta un'implementazione dell'algoritmo FedAVG (1.4) adattato al modello YOLO (You Only Look Once).
La funzione AVGForYOLO è progettata per calcolare la media ponderata dei pesi dei diversi modelli, e per poter effettuare l'aggregazione necessita di alcuni parametri, ossia: una lista dei nomi dei modelli locali forniti dai clienti, il numero del round corrente di addestramento/aggregazione ed un DataFrame, contenente informazioni sui campioni forniti dai clienti. In questo caso il numero di campioni, non sarà il numero di immagini posseduto da ogni partecipante, bensì il numero di oggetti contenuti in quelle immagini. Questa soluzione è stata applicata considerando l'eventuale presenza di client con un esiguo numero di immagini, che però potrebbero contenere un elevato numero di oggetti correttamente riportati nelle annotation, e che quindi risulterebbero proficue per il processo di training.
La funzione utilizza il modello globale del round precedente per estrarre il nome di tutti i layer e genera una copia con le stesse caratteristiche in cui ogni layer è settato a zero, generando così un modello vuoto capace di inglobare ad ogni iterazione la media ponderata dei pesi dei modelli locali. Per aggiornare il primo layer del modello vuoto, si considerano i primi layer degli n modelli di partenza e si calcola una media ponderata di quest'ultimi considerando come pesi il numero di features che sono state utilizzate per l'addestramento dei diversi modelli. Questo procedimento viene ripetuto iterativamente per ogni layer che compone il modello vuoto, in questo modo viene generato il modello Globale. Il modello aggregato risultante dell'operazione di aggregazione è infine salvato in un file utilizzando la libreria PyTorch. Infine la funzione restituisce il riferimento (path) al file salvato.

Capitolo 5

Valutazioni

Nel presente capitolo tratterò i risultati ottenuti dai vari test che ho deciso di effettuare e che reputo maggiormente significativi. Tutti i test sono stati eseguiti utilizzando "SmartFed" da me sviluppato e presentato nel capitolo 4. Di seguito alcune assunzioni già esposte in precedenza, ma che ci tengo a ricapitolare poichè necessarie per l'analisi che tratterò nei sottoparagrafi successivi:

1. Le varie entità partecipanti al sistema federato sono deployate sullo stesso HOST, per cui hanno a disposizione la stessa capacità computazionale. É stato quindi necessario gestire la condivisione delle risorse. Questa condivisione ha determinato la necessità di eseguire dei test addestrando i modelli sui diversi partecipanti in maniera asincrona, non sovraccaricando le risorse disponibili.

2. Il numero di modelli necessari per procedere con l'aggregazione viene stabilito da una soglia statica.

3. Il database utilizzato è fortemente sbilanciato e contiene dati non-IID (cfr capitolo 1 sezione 1.3.2).

4. Per tutti gli addestramenti è stato utilizzato il modello YOLO preaddestrato sul dataset COCO.

5. La gestione della GPU è in carico al Server nel CFL, mentre è onere di un'entità esterna nel DFL.

L'obbiettivo che mi ero preposto di raggiungere con questi test non era quello di ottenere la massima accuratezza possibile, bensì quello di testare "SmartFed", al fine di poter effettuare un confronto tra i risultati ottenuti attraverso il suo utilizzo e quelli che si otterrebbero mediante un addestramento classico con dataset decentralizzato.

5.1 Test addestramento classico

Al fine di avere a disposizione un metro di paragone con il quale confrontare i risultati ottenuti attraverso l'utilizzo di "SmartFed", ho eseguito innanzitutto un addestramento del modello YOLO utilizzando l'intero dataset Vis-drone. L'addestramento è stato eseguito per 100 epoche e nella tabella 5.1, vengono riportati i risultati dell'addestramento nell'epoca in cui è stata registrata la migliore accuratezza.

Tabella 5.1: Results

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
all	1610	69257	0.523	0.402	0.399	0.232
pedestrian	1610	21006	0.563	0.379	0.399	0.164
people	1610	6376	0.518	0.282	0.283	0.0977
bicycle	1610	1302	0.342	0.175	0.147	0.0561
car	1610	28074	0.737	0.787	0.789	0.509
van	1610	5771	0.507	0.413	0.405	0.272
truck	1610	2659	0.542	0.47	0.479	0.306
tricycle	1610	530	0.34	0.306	0.235	0.121
awning-tricycle	1610	599	0.41	0.249	0.222	0.124
bus	1610	2940	0.751	0.553	0.627	0.441

La tabella 5.1 riporta i risultati ottenuti dal modello YOLO addestrato sul dataset di training e testato sul dataset di test, e fornisce diverse metriche di valutazione per ciascuna classe. Ogni riga della tabella contiene i risultati del modello per una classe specifica, fornendo informazioni sulla precisione, la recall e la Precisione Media (capitolo 2 paragrafo 2.4). I valori numerici presenti in tabella stanno ad indicare le valutazioni quantitative delle prestazioni del modello riguardo lo specifico KPI (colonna) per ogni classe di oggetti. La riga colorata di verde chiaro è costituita dai risultati complessivi di tutte le classi. Le colonne presenti in tabella sono le seguenti:

- **Classe**: definisce il nome della classe a cui si riferiscono i risultati, ovvero le valutazioni quantitative;
- **Immagini**: indica il numero di immagini presenti nel dataset di ciascuna classe;
- **Etichette**: questa colonna specifica il numero di etichette (oggetti) associati alla classe nel dataset;

- P: definisce la precisione (Precision) del modello per ciascuna classe oggetto d'analisi;

- R: rappresenta la Recall del modello;

- **mAP@.5**: specifica la Precisione Media (mean Average Precision) del modello per una data classe, con soglia di IoU (Intersezione-over-Unione) pari a 0.5;

- **mAP@.5:.95**: indica la Precisione Media del modello per ogni classe, con soglia di IoU variabile da 0.5 a 0.95 (cfr paragrafo 2.4).

La tabella fornisce una panoramica dettagliata delle prestazioni del modello, consentendo quindi di valutare la sua performance.
Analizzando la metrica Precision (P) più nel dettaglio, possiamo evincere che YOLO riesce a classificare correttamente gli oggetti nel dataset di test nel 52% dei casi. Se esaminiamo nello specifico la precision di ciascuna classe, invece, possiamo notare che il modello riscontra alcune difficoltà nell'identificazione della classe corretta per oggetti come tricicli (34%) e biciclette (34%), mentre riesce a identificare con più facilità classi come le automobili (74%), i furgoni (51%), pedoni (56%) e le persone (51%). Questa maggiore accuratezza nell'identificazione della calasse di appartenenza è dovuta principalmente a due fattori:

- **Dimensione dell'oggetto all'interno dell'immagine**;

- **Numero di occorrenze maggiore nel dataset di train**.

Come possiamo osservare nella figura 5.1, le classi che presentano un numero maggiore di campioni sono le stesse in cui si rileva un valore in termini di precision molto più elevato, per l'appunto si tratta di: car, pedestrian e peopole. Inoltre a sostegno dei problemi nell'identificazione di tricicli e biciclette, ci si aspetta che le due classi siano poco visibili all'interno dell'immagine, poichè costituite da oggetti più piccoli, ragion per cui la precisione con la quale il modello è in grado di classificare e identificare gli oggetti di queste classi in maniera corretta diminuisce notevolmente. Nello specifico tricicli e biciclette presentano una mAP@0.5 rispettivamente di 23% e 15%, che per oggetti di più grandi dimensioni, anche se con meno occorrenze, può raggiungere valori ben oltre il 60% (es. bus e car).

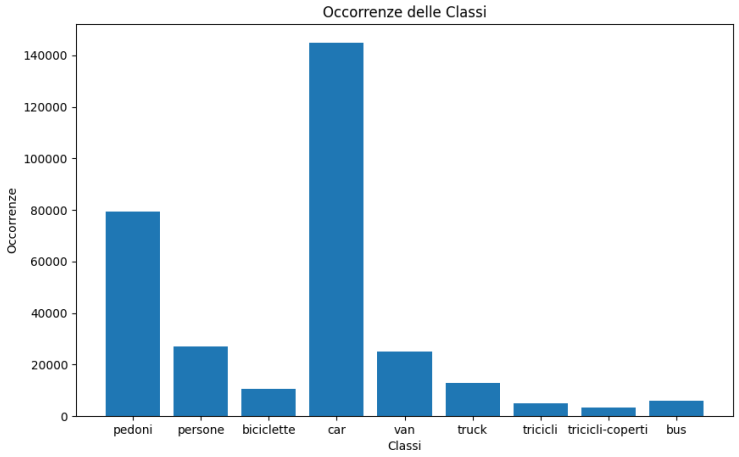

Figura 5.1: Training set

Risultato inaspettato e sorprendente è quello ottenuto dal modello nell'individuazione dei bus. Nonostante il dataset di addestramento contenesse pochi campioni per questa classe, viene identificata con una mAP@0.5 pari al 63% . Ciò può essere attribuito alla dimensione fisica distintiva del bus ed all'utilizzo di un modello preaddestrato su un database che prevedeva questa classe.
Analizzando la confusion matrics Figura 5.2 di seguito, possiamo comprendere come molto spesso oggetti che per caratteristica fisica sono di piccole dimensioni possono non essere identificate nell'immagine. Ad esempio analizzando la classe Bike si ha un 79% di falsi negativi.

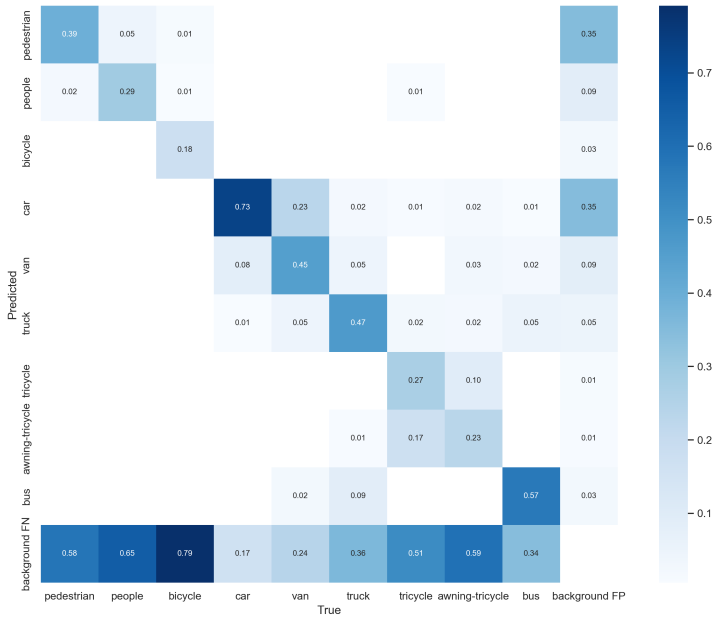

Figura 5.2: Confusion matrix train classico

Sempre esaminando la confusion matrics possiamo notare che il modello confonde le classi tricycle ed awning-tricycle. Questi errori nel riconoscimento delle due tipologie di oggetto sono imputabili alla loro somiglianza. L'errata generalizzazione del modello potrebbe essere risolta aumentando il numero di campioni per entrambe le classi, in questo modo il modello riuscirebbe ad identificare un numero maggiore di caratteri distintivi degli oggetti riuscendo quindi a raggiungere un'accuratezza maggiore. Nella Figura 5.3 è possibile vedere un'immagine in cui vengono riportate le bounding box salvate nell'annotation, mentre nella 5.4 è possibile osservare il risultato che si ottiene dall'applicazione del modello di object detection YOLO in seguito all'addestramento.

Figura 5.3: Immagine dataset di test con applicazione annotation

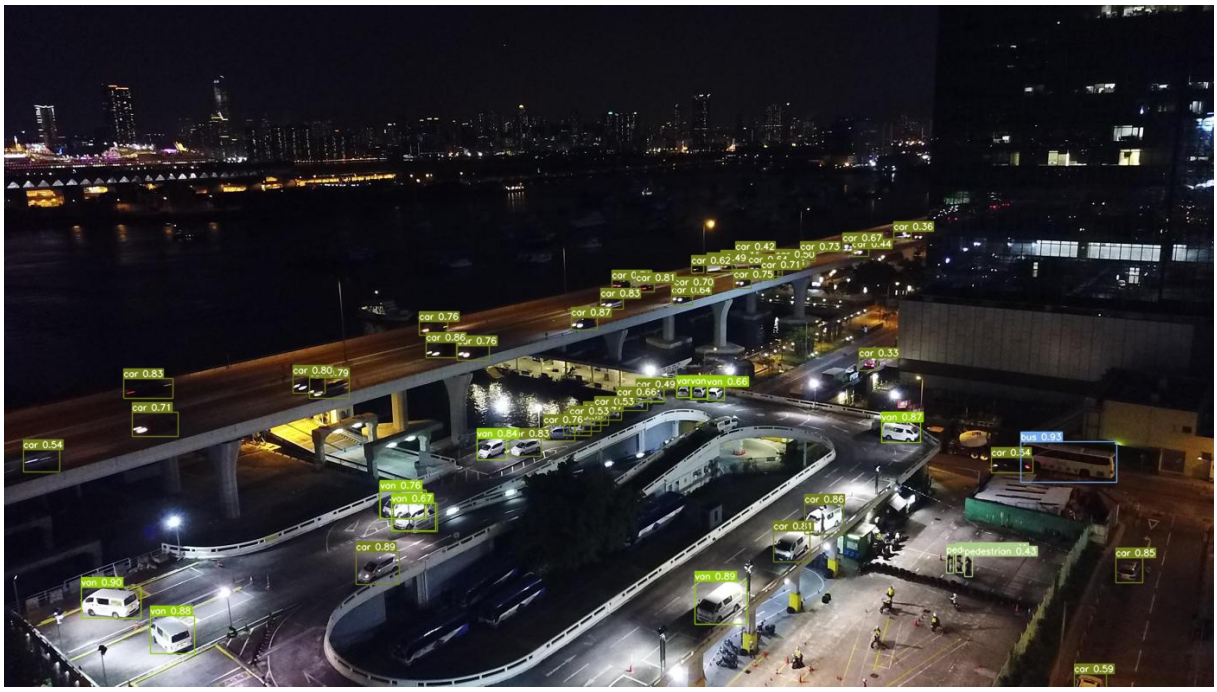

Figura 5.4: Immagine risultante dell'applicazione del modello YOLO

Come si può chiaramente notare, YOLO presenta alcuni limiti nell'identificazione di oggetti in penombra, ciò potrebbe essere dovuto all'assenza di oggetti con un'illuminazione simile nel dataset di addestramento. In più, come detto in precedenza, si evince anche la difficoltà nell'identificazione di oggetti molto piccoli, come ad esempio le persone in basso a destra.

Per concludere le considerazioni sulle prestazioni di YOLO addestrato seguendo una metodologia classica (con l'utilizzo di un dataset centralizzato), possiamo affermare che i risultati ottenuti possono essere ritenuti soddisfacenti, considerando sia lo sbilanciamento del dataset utilizzato che la presenza di dettagli molto piccoli in ciascuna immagine.

Nell'ultimo paragrafo di questo capitolo, paragonerò questi risultati, con quelli ottenuti mediante un addestramento federato. Confrontando gli esiti potremo appurare quanto affermato nella definizione al capitolo 1 sezione 1.2, ossia che uno modello addestrato seguendo un processo federato fornisce un'accuratezza simile o poco minore dello stesso modello addestrato attraverso procedura standard.

5.2 Test CFL (Centralized federated learning)

Attraverso l'implementazione del CFL (Centralized Federated Learning) presentata nel Capitolo **4, Sezione 4.1**, sono stati condotti diversi test per valutare l'efficienza e l'efficacia di questa metodologia di addestramento. Per eseguire tali esperimenti, ho dovuto suddividere il dataset Vis-drone tra i vari nodi e successivamente eseguire il processo di addestramento su ciascuno di essi. Ho inoltre fatto in modo che tutti i nodi fossero in grado di comunicare con il server centrale, il quale ha svolto il compito di aggregare i modelli provenienti dai singoli nodi. I risultati ottenuti dai migliori modelli addestrati con metodologia CLF e testati sul dataset di test sono riportati nella Tabella 5.2 di seguito:

Tabella 5.2: Risultati addestramento CFL

N. Test	Tipo	N. Part.	Soglia	Labels	P	R	mAP@.5	mAP@.5:.95
1	CFL	3	3	69257	0.482	0.375	0.361	0.206
2	CFL	3	2	69257	0.469	0.388	0.365	0.209
3	CFL	7	7	69257	0.479	0.381	0.365	0.209
4	CFL	7	5	69257	0.471	0.380	0.365	0.202

Questa tabella contiene i risultati ottenuti dall'addestramento del modello YOLO attraverso il processo di centralized federated learning, e le sue colonne stanno ad indicare:

- "N. Test": il numero del test.
- "Tipo": specifica il tipo di addestramento utilizzato per allenare il modello.
- "N. Part.": rappresenta il numero di partecipanti coinvolti nell'addestramento del modello.
- "Soglia": definisce la soglia di aggregazione utilizzata durante il processo di addestramento. L'aggregazione del modello viene eseguita nel momento in cui il server ha almeno N modelli locali da aggregare.
- "Labels": indicano il numero di etichette (oggetti) associati alle immagini presenti nel dataset utilizzato per l'addestramento.
- "P": indica la precisione (Precision) del modello ottenuta dopo l'addestramento.
- "R": rappresenta il richiamo (Recall) del modello ottenuto dopo l'addestramento.
- "mAP@.5": denota l'maP (mean Average Precision) del modello, ottenuta con una soglia di IoU pari a 0.5.

- "mAP@.5:.95": specifica la Precisione Media del modello ottenuta con una soglia di IoU variabile tre 0.5 e 0.95.

I valori numerici riportati nella tabella rappresentano le medie delle prestazioni del modello per ciascun KPI. Questa tabella offre una vista di sintesi dei risultati ottenuti dai vari test, evitando di approfondire gli esiti di ogni singola classe. Ho notato che le considerazioni che emergono dall'analisi dettagliata sarebbero identiche a quelle presentate nel capitolo precedente, rendendo la discussione ripetitiva. Pertanto, ho scelto di non includere tali dati nella presente esposizione, poiché rifletterebbero gli stessi risultati ottenuti con un addestramento convenzionale. Tuttavia, tali valutazioni preliminari forniscono un'idea iniziale sulle conclusioni che saranno presentate al termine del lavoro.

Analizzando i risultati riportati nella tabella 5.2, possiamo osservare che il processo di addestramento federato centralizzato non è significativamente influenzato dall'eventuale assenza di un worker in uno specifico round. Ad esempio, analizzando le prime due righe (test 1 e test 2) della tabella in cui il dataset è stato distribuito su 3 nodi, hanno come differenza sostanziale la soglia di aggregazione preimpostata. Nel test 1 si ha una "soglia $=$ 3 $=$ N. Part.", questo sta a significare che il server prima di procedere con l'aggregazione dei diversi modelli locali ha atteso che tutti i worker, ovvero tre, completassero l'addestramento. Nel test 2, invece, ho impostato una "soglia $=$ 2 $<$ N. Part.", quindi in questo caso l'aggregazione viene fatta anche quando il server riceve solo 2 modelli locali. Nonostante questa differenza, i risultati ottenuti sono molto simili:

- $mAP_{test1} = 36.1\% \sim mAP_{test2} = 36.5\%$
- $P_{test1} = 48.2\% \sim P_{test2} = 26.9\%$
- $R_{test1} = 37.5\% \sim R_{test2} = 46.9\%$.

Ho notato che la differenza principale tra i due test risiede nel fatto che nel secondo sono necessari più round per ottenere il miglior risultato. Ciò lo si può evincere anche dalle figure 5.5 e 5.6, dove viene mostrato l'andamento della precisione (P) del modello all'aumentare del numero di round eseguiti. Nel primo test la precision migliore viene raggiunta nel round 6, mentre nel secondo al round 12. Situazione analoga si è presentata nei test 3 e 4, inoltre, in questi test la differenza tra i vari KPI utilizzati per la valutazione del modello si riduce ulteriormente.
In sintesi, l'analisi dei risultati suggerisce che il processo di addestramento federato centralizzato è robusto rispetto alla partecipazione dei worker in quanto, l'aumento del numero di round di addestramento può compensare la mancanza di partecipanti in alcuni round. Oltre a questo è importante notare che esiste anche una relazione di compensazione tra il numero di partecipanti alla federazione e il numero di modelli utilizzati per il calcolo

del modello globale ad ogni round. Ciò significa che l'aumento dei partecipanti, rende il sistema più resistente all'eventuale assenza di alcuni di essi.

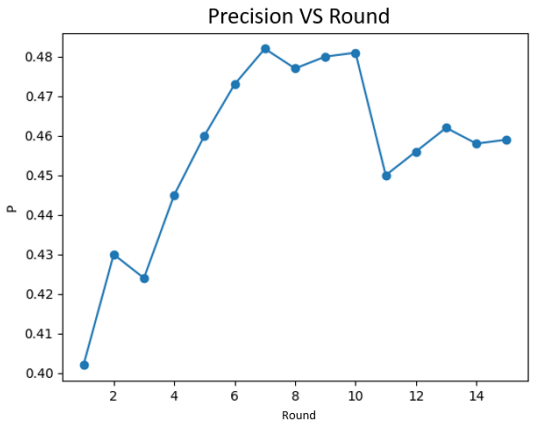

Figura 5.5: AVG Precision per Round (test 1)

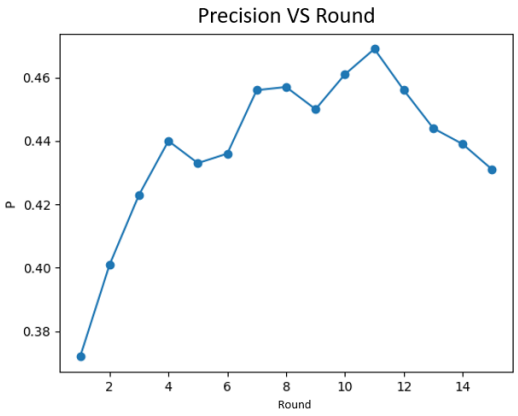

Figura 5.6: AVG Precision per Round (test 2)

Mettendo a confronto i valori della colonna "mAP@.5", possiamo osservare che i risultati variano tra 0.361 e 0.365. Questi valori indicano la capacità del modello di discriminare correttamente gli oggetti contenuti nelle immagini e di fornire un'indicazione precisa delle loro posizioni a meno di una soglia di precisione $aP = 0.5$. Più alta sarà la mAP, migliore sarà la performance del modello nel rilevamento degli oggetti. Dalla tabella, emerge che i diversi test presentano una variazione minima o inesistente nel valore di questa metrica, pertanto è possibile affermare che in tutti i test l'addestramento è riuscito a raggiungere lo stesso punto di convergenza. Questo risultato suggerisce che il sistema federato centralizzato è in grado di performare efficacemente e di mantenere prestazioni consistenti anche con la partecipazione di un numero maggiore di nodi, anche se questi potrebbero possedere un numero inferiore di campioni.
La stabilità del valore di mAP@0.5 evidenzia la robustezza del sistema federato nel mantenere la qualità del rilevamento degli oggetti, nonostante la distribuzione dell'addestramento su più partecipanti. Ciò implica che l'algoritmo `FedAVGforYOLO`, utilizzato in questi test, riesce a combinare in modo efficace le informazioni dai nodi partecipanti, senza compromettere la precisione complessiva del rilevamento.
Questi risultati dimostrano la validità del framework ("SmartFed") di apprendimento federato implementato e utilizzato, e quanto questo sia in grado di gestire con successo la collaborazione tra nodi eterogenei, fornendo una solida base per l'estensione del sistema a un numero ancora maggiore di partecipanti.
Ad ulteriore conferma delle performace che ho ottenuto attraverso l'utilizzo di questa metodologia di apprendimento, ora vi propongo il confronto tra le confusion matrix dei

deversi test ed attraverso le quali possiamo notare come in diversi scenari il risultato dell'apprendimento federato converga sempre verso lo stesso punto, sebbene, come spiegato in precedenza, con delle tempistiche in termini di round differenti.

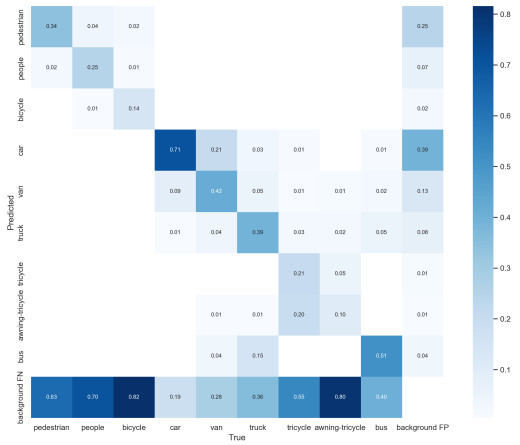

Figura 5.7: Test 1

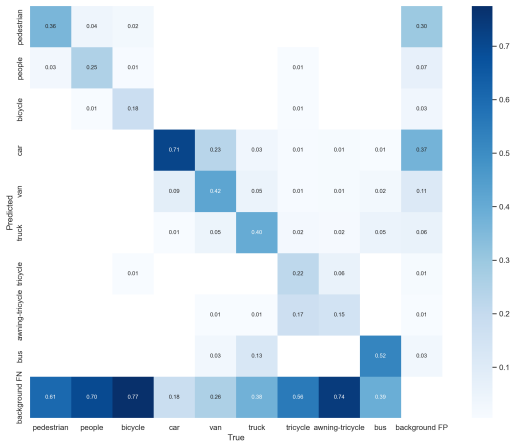

Figura 5.8: Test 2

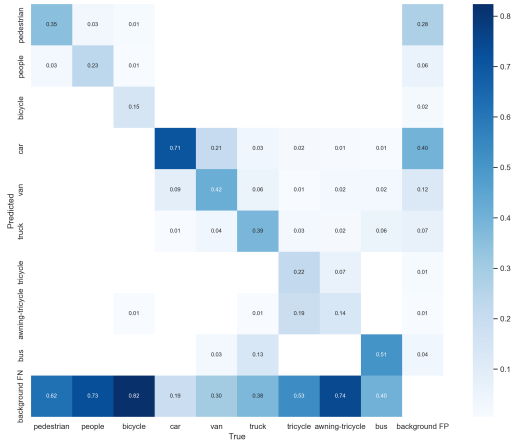

Figura 5.9: Test 3

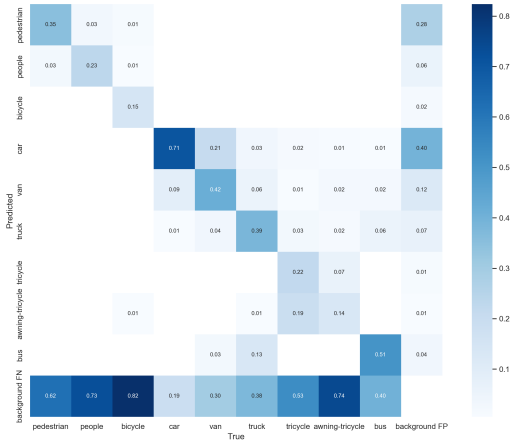

Figura 5.10: Test 4

Dall'analisi delle matrici sparse, emerge che il modello presenta alcune difficoltà nell'identificazione di alcune classi di oggetti. In particolare, si osserva un elevato numero di falsi negativi per la maggior parte delle classi, ad eccezione di quelle caratterizzate da dimensioni fisiche rilevanti (come bus, van e truck), o di quelle che presentano un ampio numero di campioni nel dataset di addestramento, che come già specificato consente di estrarre un numero sufficiente di caratteristiche per l'identificazione (come avviene ad esempio per car). Inoltre, dalla confusion matrix si evince che il modello fatica ad effettuare una distinzione tra le classi di tricicle e awning-tricicle, un risultato che è stato già evidenziato nel paragrafo 5.1, durante un addestramento "classico" con database centralizzato. Pertanto, non è possibile attribuire questo errore di classificazione all'addestramento federato.

5.3 Test DFL (no Server)

Sfruttando l'implementazione del DFL presentata nel paragrafo 4.3, sono stati effettuati gli stessi test riportati nel paragrafo precedente. I risultati del processo di addestramento federato decentralizzato sono riportati nella tabella 5.3 a seguire:

Tabella 5.3: Risultati addestramento DFL

N. Test	Tipo	N. Part.	Soglia	Labels	P	R	mAP@.5	mAP@.5:.95
1	DFL	3	3	69257	0.485	0.371	0.360	0.208
2	DFL	3	2	69257	0.469	0.390	0.362	0.207
3	DFL	7	7	69257	0.479	0.380	0.363	0.210
4	DFL	7	5	69257	0.471	0.380	0.363	0.201

Nei grafici in calce viene raffigurata la precision (P) calcolata sul modello aggregato determinato da ogni client. Al termine dell'addestramento per il test 1 viene eletto come miglior modello aggregato quello del Worker 1. Il modello in questione al round 4 registra una precisione media di 0.485 (possiamo dire che il modello è in grado di caratterizzare correttamente gli oggetti identificati all'incirca nel 50% dei casi) e un'mAP@0.5 del 36% . Mentre nel secondo test il modello con la precision migliore viene generato al tredicesimo round e sarà quello del worker 3 con precision del 47% e una mAP sempre nell'intorno del 36%.

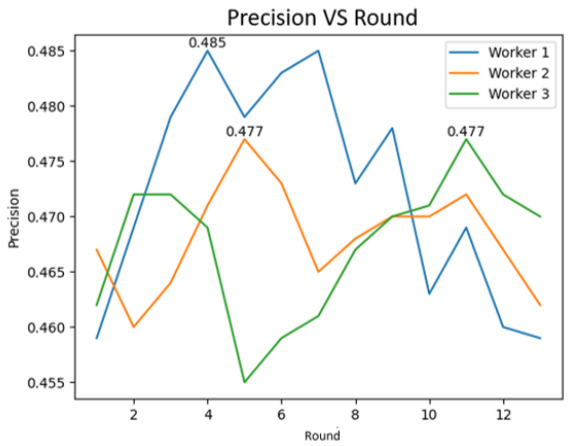

Figura 5.11: AVG Precision X Round test 1

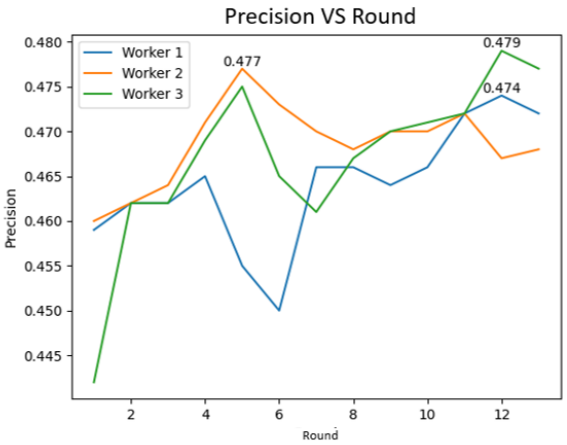

Figura 5.12: AVG Precision X Round test 2

Anche in questi test possiamo notare analizzando i risultati ottenuti, che il processo di addestramento federato decentralizzato (come per quello centralizzato) non è significativamente influenzato dall'eventuale assenza di un worker in uno specifico round. L'eventuale

assenza di alcuni partecipanti anche in questo caso sarebbe facilmente colmabile attraverso l'esecuzione di un maggiore numero di round. I risultati che si ottengono sono molto simili a quelli ottenuti nel processo centralizzato. Anche in questi esperimenti ho ottenuto una mAP@.5 che si aggira intorno al 36% ed una precisione al 47

Mettendo ora a confronto le confusion matrix dei deversi test, come negli esperimenti riportati per il CFL, possiamo notare come in diversi scenari il risultato dell'apprendimento federato converga sempre verso lo stesso punto. La situazione si conferma molto simile se non identica a quella ottenuta nel paragrafo precedete nei test presentati sul CFL.

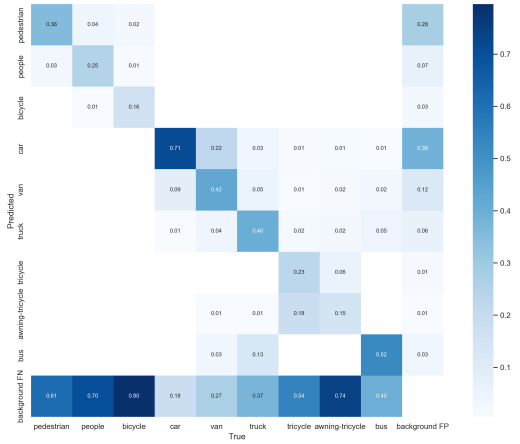

Figura 5.11: Test 1

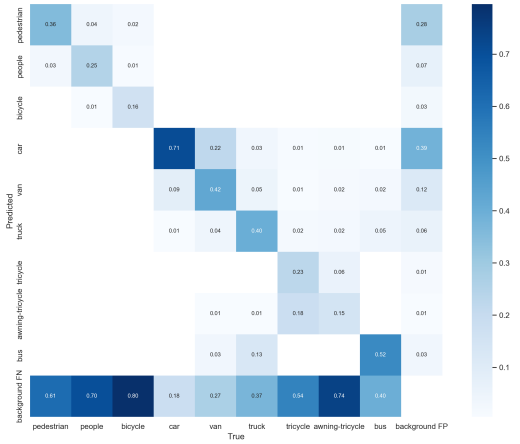

Figura 5.12: Test 2

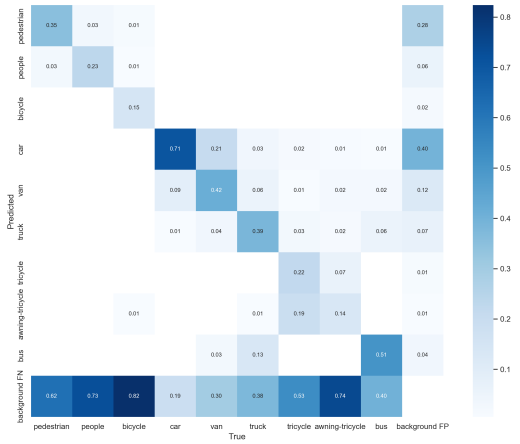

Figura 5.13: Test 3

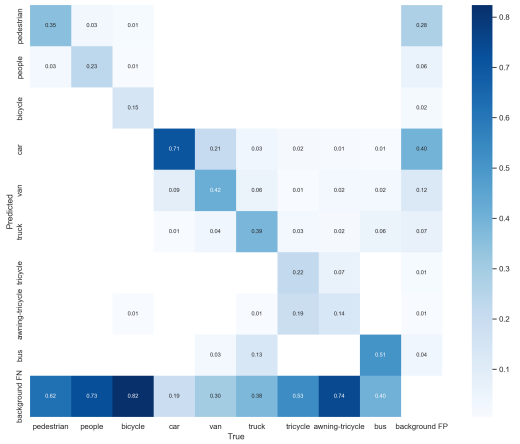

Figura 5.14: Test 4

5.4 Test a confronto

Tabella 5.4: Risultati addestramento CFL e DFL Vs Addestramento dataset centralizzato

N. Test	Tipo	N. Part.	Soglia	Labels	P	R	mAP@.5	mAP@.5:.95
1	CFL	3	3	69257	0.485	0.371	0.360	0.208
2	CFL	3	2	69257	0.469	0.390	0.362	0.207
3	CFL	7	7	69257	0.479	0.381	0.363	0.210
4	CFL	7	5	69257	0.471	0.380	0.363	0.201
5	DFL	3	3	69257	0.482	0.375	0.361	0.206
6	DFL	3	2	69257	0.469	0.388	0.365	0.209
7	DFL	7	7	69257	0.479	0.381	0.365	0.209
8	DFL	7	5	69257	0.471	0.380	0.365	0.202
0	CL	-	-	69257	0.523	0.402	0.399	0.232

La tabella in alto, riporta i risultati dei diversi test eseguiti utilizzando le diverse tipologie di addestramento: CFL (Addestramento Federato Centralizzato), DFL (Addestramento Distribuito Federato) e CL (Addestramento con Dataset Centralizzato). Ogni riga rappresenta un diverso test di addestramento effettuato con specifici parametri.

Possiamo notare che i test di addestramento CFL e DFL hanno gli stessi valori per il numero di partecipanti (N. Part.) e la soglia (Soglia), questo al fine di eseguire simulazioni il più simili possibile. Ho cercato di rendere il confronto dei risultati esplicativo, al fine di comprendere al meglio il funzionamento o meno delle due architetture. Per entrambe le tipologie di addestramento, ho utilizzato 3 e 7 partecipanti con soglie sul numero di concorrenti per la generazione del modello globale pari a 3 e 2 per quello con 3 partecipanti, mentre ho scelto 5 e 7 per i test con più nodi nella federazione, ovvero 7. Tutti i test sono stati eseguiti con lo stesso dataset di addestramento composto da 6471 immagini e testati sul dataset di test da 1610 immagini, anche in questo caso ho preso questa decisione al fine di poter confrontare i risultati.

Osservando le metriche di valutazione, possiamo notare che i valori di precisione (P), richiamo (R) e Precisione Media (mAP@0.5 e mAP@0.5:0.95) variano leggermente tra CFL e DFL, ma non mostrano una differenza significativa. Quindi possiamo affermare che l'utilizzo dei due sistemi di federated learning non determina miglioramenti o peggioramenti sulla qualità del modello addestrato.
Quanto appena specificato, conferma quanto avevo anticipato nel paragrafo 1.2.2 del capitolo 1, infatti, dal momento che i risultati ottenuti non si differenziano tra le due tipologie di apprendimento federato, DFL può essere un ottimo sostituto di CFL. Ciò comporta

una serie di vantaggi, i cui principali sono la non dipendenza dal server centrale e la possibilità di aggiungere un numero elevato di nodi (scalabilità).
In confronto ai due metodi appena trattati, l'addestramento con dataset centralizzato (CL) ha riportato valori più elevati per tutte le metriche di valutazione. La precisione (P), il richiamo (R) e la Precisione Media (mAP@0.5 e mAP@0.5:0.95), infatti, risultano essere maggiori in questo caso di addestramento.
I risultati del CL suggeriscono che l'addestramento con dataset centralizzato potrebbe produrre prestazioni migliori rispetto all'addestramento federato centralizzato e distribuito. Tuttavia quando si valutano le diverse tipologie di addestramento è importante considerare altri fattori come la scalabilità, la privacy e l'efficienza computazionale. In più le differenze che si ottengono tra l'addestramento con le due tipologie di FL ed il CL sono esigue e quindi in linea con la definizione presentata nel capitolo 1 sezione 2.
Inoltre, nelle immagini seguenti si possono osservare alcuni casi in cui i modelli addestrati mediante l'utilizzo del FL dimostrano una migliore capacità di generalizzare le caratteristiche delle classi, consentendo l'individuazione di un maggior numero di oggetti. Nell'immagine 5.17, che rappresenta il risultato dell'applicazione del modello addestrato con dataset centralizzato, si può vedere che il modello non riesce ad individuare nè un autobus in penombra, nè alcune persone nell'angolo in basso a destra, ma tuttavia è in grado di individuare con buona precisione alcune persone molto piccole nello stesso angolo.

Figura 5.15: Applicazzione modello CL

Nella figura 5.18, che espone il risultato dell'utilizzo del modello addestrato tramite il federated learning decentralizzato, si può notare che l'algoritmo di deep learning riesce

a rilevare anche l'autobus in penombra non individuato dal modello addestrato con il metodo classico, oltre a identificare anche alcune persone su di una moto, rimaste invece non rilevate nel modello precedente. Va però condiviso anche il fatto che sono stati riscontrati alcuni casi in cui le prestazioni erano inferiori, come ad esempio l'assenza di rilevamento delle persone nell'angolo in basso a destra dell'immagine.

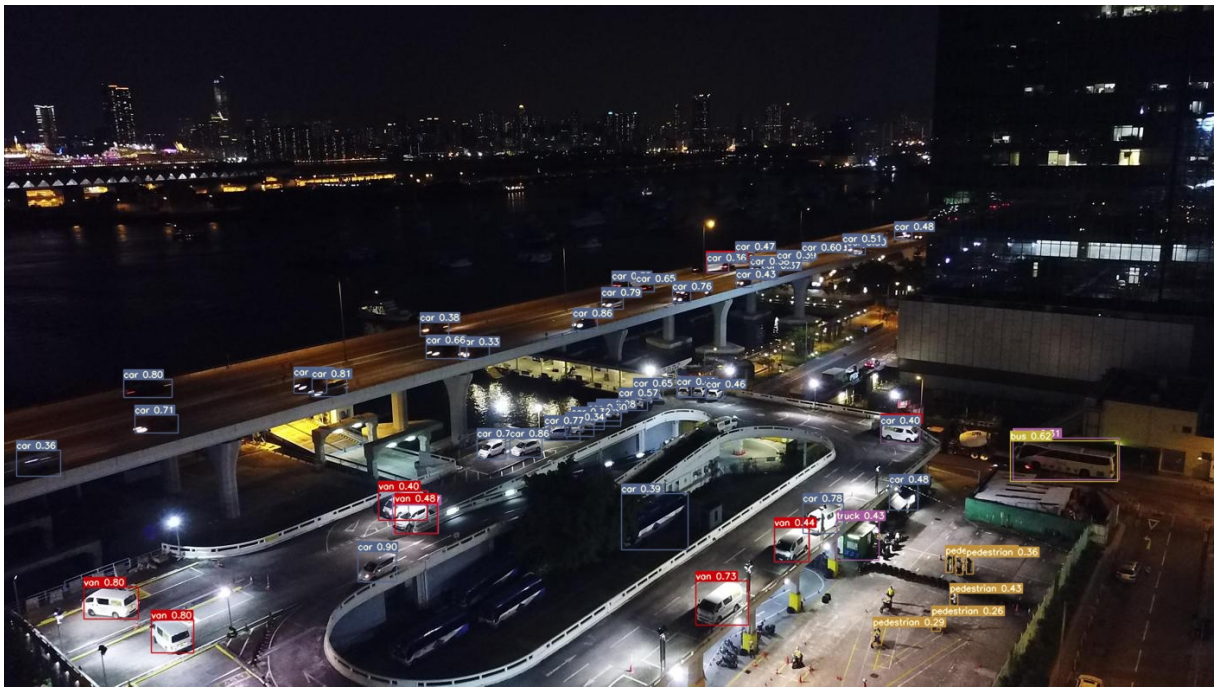

Figura 5.16: Applicazzione modello DFL

Figura 5.17: Applicazzione modello CFL

Conclusioni

Nel mio lavoro di tesi ho deciso di esaminare in modo dettagliato, per mezzo di varie analisi e test, i vantaggi e gli svantaggi riguardanti l'utilizzo dell'apprendimento federato nell'addestramento dei modelli di deep learning.
Nello specifico, ho dedicato particolare attenzione all'addestramento del modello di object detection utilizzando il modello YOLO (You Only Look Once), che a differenza di altri approcci i quali richiedono una scansione multipla dell'immagine, adotta un metodo unico che consente di analizzare l'immagine una sola volta, offrendo risultati rapidi ed efficienti.

Per condurre l'addestramento del modello, ho utilizzato il database Vis-drone, composto da un ampio set di immagini acquisite da diversi dispositivi aerei in differenti situazioni ed ambienti. Al fine di esplorare a pieno le potenzialità dell'apprendimento federato, ho deciso anche di sviluppare da zero entrambe le possibili architetture di FL, ovvero quella centralizzata e quella decentralizzata. A differenza delle implementazioni presenti in letteratura, ho scelto di creare un nuovo framework personalizzato che consentisse un addestramento semplice, veloce e indipendente dal modello specifico oggetto dell'addestramento, al quale ho dato il nome di "SmartFed".

Dopo aver presentato tutte le implementazioni sviluppate, ho proceduto a confrontare i risultati ottenuti da ogni modalità di addestramento, focalizzando l'attenzione soprattutto sui risultati derivanti dall'addestramento "classico", basato su un database centralizzato, e quelli ricavati tramite l'addestramento federato, per entrambe le sue topologie proposte. L'esito ottenuto da queste comparazioni porta ad una conclusione significativa: il federated learning si configura come un paradigma innovativo e promettente per l'addestramento dei modelli di deep learning.
Nonostante si sia riscontrata una leggera riduzione delle prestazioni rispetto all'addestramento classico, è fondamentale considerare i vantaggi intrinseci che derivano dall'utilizzo di questa metodologia. In particolare, l'apprendimento federato garantisce la privacy di ogni partecipante, consentendo di mantenere i dati sensibili all'interno dei rispettivi dispositivi. Inoltre, offre la possibilità di distribuire il carico computazionale su diversi nodi, migliorando l'efficienza e la scalabilità complessiva del sistema di addestramento.

In definitiva, la mia ricerca ha confermato che l'apprendimento federato rappresenta una soluzione molto valida per l'addestramento di modelli di deep learning. Pur presentando alcune limitazioni in termini di prestazioni, i suoi numerosi vantaggi, come la privacy dei dati e la distribuzione del carico computazionale, lo rendono un approccio interessante e promettente per il futuro dello sviluppo di modelli di machine learning.

Viviamo in un mondo in continua evoluzione, in cui la ricerca e il progresso sono indirizzati a migliorare la vita umana ed a semplificare i processi decisionali grazie all'intelligenza artificiale.

Fino a pochi anni fa, ci si concentrava sullo sviluppo dei primi percettroni, una forma embrionale di reti neurali. Successivamente, si sono sviluppati modelli sempre più complessi, composti da strati multipli, nel tentativo di replicare il processo decisionale umano. Ora, stiamo assistendo alla creazione di un vero e proprio "sistema sociale", che permette lo scambio di informazioni mirato a migliorare l'apprendimento di questi sistemi.

Il federated learning rappresenta un importante passo avanti nell'intelligenza artificiale, consentendo di sfruttare la conoscenza collettiva per migliorare l'efficacia dei modelli di apprendimento automatico. Ciò apre nuove possibilità per l'innovazione ed il progresso, contribuendo a risolvere problemi complessi in modo collaborativo e responsabile.

References

[1] A. Rahman, M. S. Hossain, G. Muhammad, D. Kundu, T. Debnath, M. Rahman, M. S. I. Khan, P. Tiwari, and S. S. Band, "Federated learning-based AI approaches in smart healthcare: concepts, taxonomies, challenges and open issues" Cluster Computing, Aug. 2022. DOI: 10.1007/s10586-022-03658-4.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," ACM Trans. Intell. Syst. Technol., vol. 10, no. 2, pp. 12:1-12:19, Feb. 2019.

[3] Jain, "Federated Learning for Commercial Image Classification," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023. DOI: 10.1109/WACV.2023.00227.

[4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," Google, Inc., Jan. 2023. https://arxiv.org/pdf/1602.05629.pdf.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, 2015. https://arxiv.org/pdf/1502.03167.pdf

[6] M. AprilPyone and H. Kiya, "Privacy-preserving image classification using an isotropic network," IEEE MultiMedia, vol. 29, no. 2, pp. 23-33, 2022. https://arxiv.org/pdf/2204.07707.pdf

[7] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, Vikas Chandra "Federated Learning with Non-IID Data" https://doi.org/10.48550/arXiv.1806.00582

[8] *Jayanta Kumar ,Basu, Debnath Bhattacharyya, Tai-hoon Kim, "Use of Artificial Neural Network in Pattern Recognition," International Journal of Software Engineering and Its Applications, Vol. 4, No. 2, April 2010* `https://www.researchgate.net/profile/Debnath-Bhattacharyya/publication/228566377_Use_of_Artificial_Neural_Network_in_Pattern_Recognition/links/09e4150ff1c2e41705000000/Use-of-Artificial-Neural-Network-in-Pattern-Recognition.pdf`

[9] Redazione Tech, ''Ecco come il deep learning aiuta i computer a rilevare gli oggetti", 21 Febbraio 2023 https://www.techgeneration.it/ecco-come-il-deep-learning-aiuta-i-computer-a-rilevare-gli-oggetti/#YOLO

[10] https://medium.datadriveninvestor.com/an-overview-of-federated-learning-8a1a62b0600d

[11] Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong,N., Ramage, D., and Beaufays, F. Applied federated learning: Improving google keyboard query suggestions. arXiv preprint 1812.02903, 2018.https://arxiv.org/abs/1812.02903

[12] Y. Liu, J. J. Q. Yu, J. Kang, D. Niyato and S. Zhang, "Privacy-Preserving Traffic Flow Prediction: A Federated Learning Approach," in IEEE Internet of Things Journal, vol. 7, no. 8, pp. 7751-7763, Aug. 2020, doi: 10.1109/JIOT.2020.2991401.

[13] T. Li, A. K. Sahu, A. Talwalkar and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," https://arxiv.org/pdf/1908.07873.pdf

[14] Osama Shahid, Seyedamin Pouriyeh, Reza M. Parizi, Quan Z. Sheng, Gautam Srivastava, Liang Zhao "Communication Efficiency in Federated Learning: Achievements and Challenges" https://doi.org/10.48550/arXiv.2107.10996

[15] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," arXiv preprint arXiv:1804.08333, 2018 https://arxiv.org/pdf/1804.08333.pdf

[16] Artur Back de Luca, Guojun Zhang, Xi Chen, Yaoliang Yu "Mitigating Data Heterogeneity in Federated Learning with Data Augmentation" https://doi.org/10.48550/arXiv.2206.09979

[17] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Xuyun Zhang "Source Inference Attacks in Federated Learning"https://doi.org/10.48550/arXiv.2109.05659

[18] Federated Learning meets Homomorphic Encryption https://research.ibm.com/blog/federated-learning-homomorphic-encryption

[19] Gabriele Tolomei, Edoardo Gabrielli, Dimitri Belli, Vittorio Miori "A Byzantine-Resilient Aggregation Scheme for Federated Learning via Matrix Autoregression on Client Updates"https://doi.org/10.48550/arXiv.1610.05492

[20] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi "You Only Look Once: Unified, Real-Time Object Detection" https://doi.org/10.48550/arXiv.1506.02640

[21] Hosna, A., Merry, E., Gyalmo, J. et al. Transfer learning: a friendly introduction. J Big Data 9, 102 (2022). https://doi.org/10.1186/s40537-022-00652-w

[22] @articlezhuvisdrone2018, title=Vision Meets Drones: A Challenge, author=Zhu, Pengfei and Wen, Longyin and Bian, Xiao and Haibin, Ling and Hu, Qinghua, journal=arXiv preprint:1804.07437, year=2018

[23] Flower Documentation https://flower.dev/docs/

[24] PYSYFT https://blog.openmined.org/tag/pysyft/

[25] TensorFlow Federated: Machine Learning su dati decentralizzati https://www.tensorflow.org/federated?hl=it