

UNIVERSITÀ DI SIENA

1240

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE E
SCIENZE MATEMATICHE

Corso di Laurea in
Ingegneria informatica e dell'informazione

Classificazione su base audio del traffico con
algoritmi di TinyML su dispositivi embedded

Relatore:
Prof.ssa Ada Fort

Candidato:
Francesco Maccantelli

Anno Accademico 2021-2022

Indice

Introduzione	3
1 Componenti hardware e firmware	8
1.1 Arduino Nano 33 BLE Sense	8
1.1.1 Microfono MP34DT05	8
1.1.2 Script Arduino	9
2 Tecniche di analisi	13
2.1 Registrazione dei campioni	13
2.2 Software di analisi	15
2.2.1 Classificazione dei campioni	16
2.2.2 Scelta valori soglia	24
2.2.3 Export campioni audio	25
3 EdgeImpulse	27
3.1 EdgeImpulse	27
3.2 Caso di studio	29
3.2.1 1° Spot di rilevazione	29
3.2.2 2° Spot di rilevazione	31
3.3 Esportazione del Modello	34
Conclusioni	36

Introduzione

In questo progetto di tesi è stato ideato un sistema in grado di sfruttare tecniche di TinyML (Tiny Machine Learning) per la realizzazione di un classificatore che tramite l'analisi di un segnale audio potesse stimare il livello di congestione stradale.

Per raggiungere l'obiettivo proposto inizialmente è stato analizzato il problema, cercando conseguentemente la letteratura già presente su questo argomento, individuando i passaggi fondamentali ed infine le tecniche per conseguire lo scopo proposto.

L'idea alla base di questo progetto di tesi è quella di realizzare un dispositivo, che sfruttando l'analisi di un segnale audio generato da un microfono, possa stimare il livello di congestione stradale, il tutto basato su tecniche di TinyML eseguite su dispositivi embedded.

Questo approccio permetterebbe di ridurre molto i costi rispetto ad altri sistemi usati per simili scopi (analisi tramite videocamere, sensori stradali), inoltre questo sistema permetterebbe anche un'installazione non invasiva, a differenza di quella necessaria per il collocamento di sensori al di sotto del manto stradale o con specifiche necessità di un corretto posizionamento di una videocamera tale da avere una corretta visuale sulla strada.

Oltre al risparmio economico per la realizzazione di questo sistema, rispetto a metodi più classici, tale progetto di tesi è stato sviluppato con ottica di un futuro sviluppo e miglioramento tale da poter rilevare condizioni ambientali quali: inquinamento dell'aria, presenza di polveri sottili o altri componenti usando il medesimo approccio, beneficiando di tutti i lati positivi che l'utilizzo del TinyML apporta.

Proprio per queste necessità e per altre caratteristiche, che verranno illustrate successivamente, è stato deciso di utilizzare un Arduino Nano 33 BLE

Sense [1], come dispositivo atto alle registrazione audio, grazie al microfono già integrato sulla board e dotato di funzionalità per la classificazione tramite algoritmi di TimyML.

Training set

La prima fase di questo progetto si è basata sulla creazione di un dataset che potesse permettere il corretto addestramento della rete neurale. La creazione del training set è una fase fondamentale, dalla quale dipende l'affidabilità dell'algoritmo che viene generato. Per questo motivo sono state adoperate varie tecniche, che verranno illustrate successivamente.

EdgeImpulse

Una volta realizzato un dataset che sia sufficientemente popolato di dati, suddiviso equamente nelle varie classi, è stata usata la piattaforma EdgeImpulse,¹ la quale tramite una procedura guidata, permette di sfruttare l'apprendimento automatico per la realizzazione di un modello matematico, nel caso in questione atto alla classificazione del livello di traffico presente in una strada.

La particolarità della piattaforma EdgeImpulse è che permette l'utilizzo del machine learning (ML) per la generazione di un modello addestrato in modo automatico, che può essere eseguito su dispositivi embedded come l'Arduino Nano 33 BLE Sense [2]. La generazione del modello può essere basata su un'ampia scelta di tipologia di dati. I formati accettati dalla piattaforma sono vari: WAV, JPG, PNG, CBOR, CSV, JSON, MP4 e AVI.

TinyML

Parte fondante di questo lavoro di tesi è il TinyML. Possiamo tradurre TinyML con *Tiny Machine Learning*, quindi un'applicazione dell'apprendimento automatico effettuata in miniatura. Difatti il TinyML si promette di portare il mondo del ML su dispositivi IoT (Internet Of Things), dispositivi che fino ad oggi se avessero voluto usare le capacità del ML si sarebbero dovuti appoggiare probabilmente su sistemi cloud.

¹www.edgeimpulse.com

A causa delle grandi quantità di dati che generalmente sono generate dai dispositivi IoT, la velocità di trasmissione degli stessi rischia di diventare un vero problema nei sistemi basati sul cloud, soprattutto per applicazioni in cui la connettività non può essere data per scontato. Per questo e per altri motivi quali: abbassamento della latenza, utilizzo della banda strettamente necessaria, incremento della sicurezza e della privacy, riduzione dei costi e aumento dell'efficienza energetica, il TinyML si promette di portare gli algoritmi di ML direttamente sul nodo IoT offrendo così una moltitudine di nuove opportunità, oltre che nuove sfide da affrontare.

Definizione di TinyML

Volendo definire il TinyML in modo più formale possiamo dire che definiamo TinyML come la pratica di eseguire modelli di ML su microcontrollori² ultra low power ($< 1 \text{ mW}$). Uno schema del flusso di lavoro del TinyML è presente nella figura 1.

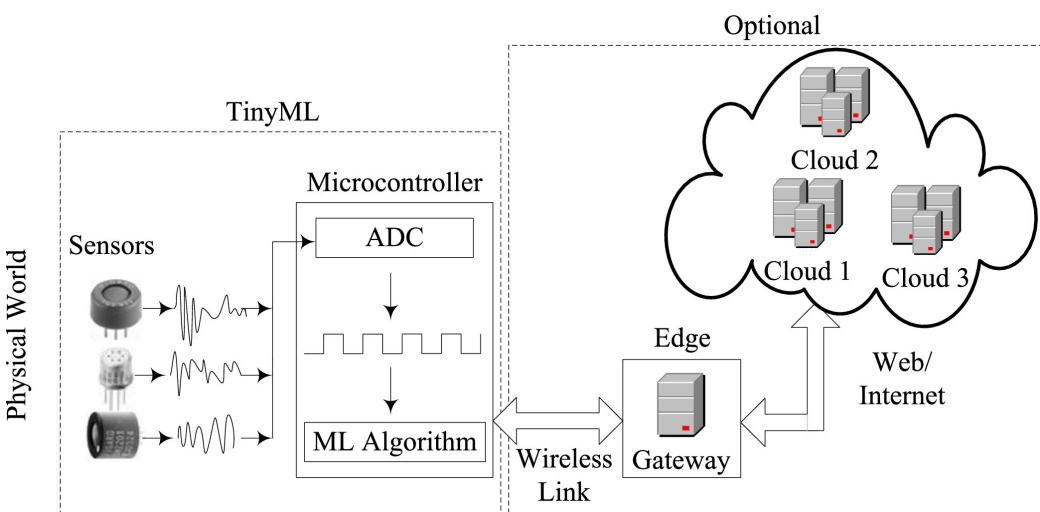


Figura 1: TinyML flusso di lavoro nel'IoT [4].

Tramite il TinyML il singolo dispositivo IoT diventa “intelligente”, offrendo così abilità di analisi dei dati, accelerando il processo decisionale e

²In elettronica digitale il microcontrollore è un dispositivo elettronico integrato su singolo chip, nato come evoluzione alternativa al Microprocessore ed utilizzato generalmente in sistemi embedded, ovvero per applicazioni specifiche di controllo digitale [3].

permettendone l'utilizzo anche in casi in cui una infrastruttura IoT standard non sia realizzabile [4].

Capitolo 1

Componenti hardware e firmware

1.1 Arduino Nano 33 BLE Sense

Il dispositivo principale usato in questo progetto di tesi è stato l'Arduino Nano 33 BLE Sense [1]. Le motivazioni che hanno portato a scegliere proprio questo dispositivo in particolare sono varie, ma le principali sono:

- Compatibilità con l'applicativo EdgeImpulse e quindi con TensorFlow Lite;
- Microfono già integrato sulla board;
- Basso costo;
- Bassi consumi;
- Facilità di programmazione.

1.1.1 Microfono MP34DT05

Il microfono che è montato sulla board Arduino è un MP34DT05 [5]. Tale dispositivo è perfetto per questo progetto in quanto si tratta di un microfono omnidirezionale, in grado quindi di ricevere il segnale sonoro da ogni angolazione, senza che esso risulti distorto. Tale caratteristica è estremamente utile

nel contesto di utilizzo di questo progetto di tesi, in quanto il dispositivo finale, atto alla determinazione del livello di congestione stradale, può essere posizionato indiscriminatamente in ogni direzione, purché sufficientemente vicino alla strada da analizzare.

Il sensore in questione è un microfono digitale MEMS (Micro Electro Mechanical Systems) in grado di convertire le onde di pressione acustiche in un segnale digitale. Le parti fondamentali di questo dispositivo sono un elemento capacitivo in grado di convertire il segnale acustico in un segnale elettrico, in secondo luogo abbiamo un amplificatore ed infine il modulatore PDM (Pulse Duration Modulation). Il modulatore PDM è in grado di convertire il segnale analogico in un segnale a modulazione d'impulso. Il segnale di clock in ingresso permette di controllare la modulazione, nel nostro caso la frequenza di campionamento del microfono è 16 kHz. Dal momento in cui il modulatore PDM viene attivato, il segnale audio viene processato ed i dati campionati vengono salvati in un buffer [6].

La modulazione PDM è una modulazione a singolo bit ad alta frequenza, una caratteristica di questa modulazione è che integra la tecnica di *noise shaping*¹, la quale permette di avere un rumore nella banda audio relativamente basso [7].

La libreria Arduino che si interfaccia con il microfono presente sulla board “*PDM.h*”, effettua una conversione dalla PDM (la modulazione del microfono) a PCM (Pulse Code Modulation). Nella PCM il segnale audio viene rappresentato da una serie di campioni, nel caso trattato in questione si tratta di campioni a 16 bit: i campioni convertiti in modulazione PCM vengono instradati dalla libreria in modo tale da essere accessibili e disponibili per il trasferimento, per poi essere salvati come file audio.

1.1.2 Script Arduino

Lo scopo di questo script Arduino è quello di permettere il trasferimento dei campioni audio registrati dal microfono dell’Arduino al computer, dove successivamente verranno categorizzati in base al traffico ed infine usati per l’addestramento della rete neurale.

¹Tecnica usata per la minimizzazione dell’errore di quantizzazione.

Lo script “PDM_sender_1.0.ino” [8] è stato creato per il trasferimento dei campioni audio da Arduino al computer, per il loro salvataggio e la creazione del dataset. Tale script si basa sull’esempio di libreria *PDMSerialPlotter* della libreria “PDM.h”. È stato però variato il sistema con cui lo script trasmette i campioni. Nell’esempio di libreria i campioni vengono inviati tramite la funzione *Serial.println()*: questa funzione però prima di inviare i dati tramite porta seriale, li converte in ASCII. Questo metodo non permetteva di inviare i 16000 campioni al secondo generati dal microfono; riusciva a trasferirne solo $\simeq 6000$ / s. Visto che la velocità di trasmissione non era sufficientemente veloce a trasferire in tempo reale i campioni del microfono inizialmente è stato deciso di salvare i campioni in una sezione dedicata della memoria del dispositivo, per poi trasferirli successivamente alla velocità di $\simeq 6000$ / s. Fin da subito questo sistema non si è mostrato ottimale in quanto era limitato sia nella velocità sia nella quantità di campioni che si potevano salvare. La quantità massima di campioni era direttamente proporzionale alla memoria che era possibile occupare. Disponendo di 256 kB ed essendo ogni campione audio composto da 2 Byte e prodotti con una frequenza di 16000 Hz questo avrebbe voluto dire che anche occupando il 100% della memoria sarebbe stato possibile salvare solamente 8 secondi di audio sarebbero stati necessari circa 42 secondi per il trasferimento verso il computer.

È stato concluso che questa strada non era percorribile. La soluzione è stata quella di usare la funzione *SerialUSB.write()* che non converte i dati in ASCII ed inoltre permette l’invio di array di byte. La difficoltà incontrata per implementare questa funzione è stata quella che i dati salvati dal microfono erano *uint16_t* quindi dati a 2 byte, mentre la funzione *SerialUSB.write()* permette l’invio di un solo byte alla volta. Quindi si è resa necessaria una conversione dell’array nel quale il microfono salvava i dati del campionamento da 2 byte ad 1 byte. Questa conversione è stata effettuata semplicemente eseguendo il codice 1.1 in un ciclo *for* che iterasse su tutto l’array iniziale.

Codice 1.1: Conversione da 16 a 8 bit.

```
sampleBuffer_8bit[sb_index] = '\n';
sb_index++;
sampleBuffer_8bit[sb_index] = (sampleBuffer[i] >> 8) & 0xFF;
```

```
sb_index++;
sampleBuffer_8bit[sb_index] = (sampleBuffer[i] & 0xFF);
sb_index++;
```

In questo modo l'array *sampleBuffer_8bit* sarà la versione a 8 bit dell'array *sampleBuffer*, dove ogni elemento a 16 bit del buffer *sampleBuffer* è rappresentato con due elementi a 8 bit nel buffer *sampleBuffer_8bit*. È stato aggiunto anche un carattere speciale '\n' che permettesse di separare i singoli campioni. Una volta trasferiti i campioni audio, dovranno essere ricomposti per tornare a 16 bit. Per il trasferimento tramite porta USB è stato sviluppato il codice 1.2.

Codice 1.2: Trasmissione dati.

```
if (sb_index >= 1536){
    SerialUSB.write(sampleBuffer_8bit,sb_index);
    sb_index=0;
}
```

Quando l'indice *sb_index* raggiunge il valore di 1536, viene chiamata la funzione *SerialUSB.write()* che permette l'invio di tutto il buffer a 8 bit in una volta sola. Facendo così è stato possibile raggiungere la velocità 16000 [campioni] / s, permettendo un salvataggio in tempo reale dell'audio del microfono.

Capitolo 2

Tecniche di analisi

Parte fondamentale di questo progetto è l'acquisizione e la successiva analisi dei campioni audio e video per la creazione del dataset per l'addestramento della rete neurale. Per fare questo procedimento è stato sviluppato uno script Python che permettesse la cattura dei campioni audio dall'Arduino e la cattura del video dalla telecamera in modo sincronizzato.

Dopo l'acquisizione, tutti i campioni vengono processati tramite un secondo script Python che permette di classificare ogni campione, assegnando così un livello di traffico rilevato sia al video che al campione audio associato.

Terminata la parte di analisi tutti i campioni audio, adesso correttamente categorizzati, vengono caricati sulla piattaforma EdgeImpulse per permettere la creazione della rete neurale che si occuperà dell'analisi del segnale audio per rilevare il livello di traffico in tempo reale.

2.1 Registrazione dei campioni

La prima fase della creazione del dataset è l'acquisizione dei dati (audio, video) che permetteranno una categorizzazione del traffico per l'addestramento della rete neurale.

Per la registrazione dei campioni audio e video in modo sincronizzato è stato sviluppato uno script Python “record_data.py” [9] che permettesse l'esecuzione in parallelo di altri due script Python (rec_audio.py [10], rec_video.py [11]), rispettivamente per la registrazione audio tramite Arduino Nano 33

BLE Sense e per la registrazione del video tramite una webcam collegata al computer.

Lo script “record_data.py” permette la creazione di un file bash appositamente configurato per l’esecuzione in contemporanea dei due script atti alla registrazione audio e video. Tramite questo script è possibile andare a definire tutti i parametri delle registrazioni audio e video, come la lunghezza delle registrazioni, gli FPS (Frame Per Secondo), la porta seriale dove vengono ricevuti i dati dall’Arduino ed un *timestamp* che permette di identificare e correlare in modo univoco i due file. In questo caso il *timestamp* ha un formato *AnnoMeseGiorno-OraMinutoSecondo*: in questo modo potrà esserci una sola coppia di file (audio, video) che vengono generati in uno specifico istante di tempo.

Una volta creato il file bash, viene eseguito e conseguentemente vengono eseguiti gli altri due script. Lo script adibito alla registrazione video “rec_video.py” [II] utilizza la libreria *OpenCV* per gestire il flusso video della webcam e salvarlo in un file nominato “*timestamp.avi*” in modo da essere correlabile al file audio che verrà generato in contemporanea.

Contemporaneamente viene eseguito lo script “rec_audio.py” [IO], che permette la lettura tramite porta seriale dei dati relativi al segnale audio inviati dall’Arduino Nano 33 BLE Sense sfruttando lo script “PDM_sender_1.0.ino” [8]. Lo script “rec_audio.py” sfrutta la libreria *serial* per leggere i campioni inviati dall’Arduino. I dati che vengono ricevuti dall’Arduino necessitano di una conversione prima di poterli salvare in quanto sono dati a 8 bit mentre i campioni originali sono a 16 bit. Per effettuare questa conversione viene il codice 2.1:

Codice 2.1: Lettura tramite porta seriale e conversione a 16 bit.

```
for x in range(samples):
    ser.read_until()
    cc1 = ser.read(2)
    result_ += str(int.from_bytes(cc1, "big"))+", "
```

Difatti conoscendo il numero di samples che vogliamo salvare (secondi di audio * 16000), con la funzione *.read_until()* viene letta la porta seriale fintanto che non viene ricevuto il carattere ‘\n’, che è anche il carattere di separazione

che è stato utilizzato per suddividere i campioni a 16 bit. Successivamente tramite la funzione `.read(2)` vengono letti 2 byte (16 bit) dalla porta seriale, che corrispondono ad un campione audio che viene convertito nuovamente in un campione a 16 bit per poi essere salvato in un file “`timestamp.json`” correttamente formattato per il successivo caricamento su EdgeImpulse. In tale formattazione vengono specificate informazioni come la frequenza di campionamento, il dispositivo utilizzato oltre che altri parametri per una corretta lettura da parte di EdgeImpulse.

2.2 Software di analisi

Una volta completata la fase di acquisizione dei dati per la creazione del dataset, è necessario andare ad analizzare i dati acquisiti e quindi categorizzare tutti i campioni, suddividendoli nelle varie classi che il dispositivo finale dovrà quindi identificare.

Per questo progetto di tesi si è scelto di creare 3 classi: assente, presente, intenso. In questo modo sarà possibile identificare il livello di congestione stradale con 3 livelli di intensità, dal più basso *Assente* al più alto *Intenso*. Questi parametri però non possono essere univocamente determinati, in quanto dipendono da numerosi fattori, quali: tipologia della strada che andiamo ad analizzare, conformazione della stessa ed altri parametri. Per questi motivi i livelli di traffico associati alle varie classi sono parametri che dovranno essere valutati e scelti di volta in volta, per ogni spot di rilevazione. Basandosi sulle rilevazioni effettuate sarà necessario associare il livello più alto (*Intenso*) al livello di traffico più elevato, il valore *Presente* ad un valore medio di traffico fino ad associare il valore *Assente* alle rilevazioni in cui il traffico non è presente.

Per effettuare il processo di classificazione è stato sviluppato uno script Python “`main_analisi.py`” [12]. Tale script permette di svolgere varie funzioni tra cui:

- Classificazione campioni;
- Visualizzazione grafici delle classificazioni;

- Esportazione campioni suddivisi per classi pronti per il caricamento su EdgeImpulse;
- Altre funzioni secondarie, utili alla configurazione ed al controllo della classificazione (verranno trattate successivamente).

2.2.1 Classificazione dei campioni

La prima funzione del software di analisi è *video_analyze()*, che permette la classificazione dei campioni. Tramite questa funzione è possibile andare ad analizzare tutti i campioni video salvati relativi a una rilevazione, per associare ad ogni video un *indice di traffico* (t_i).

Il valore t_i che viene calcolato permette di stimare il livello di traffico per ogni campione video, associando poi tale valore anche al campione audio, in modo tale che alla fine di questo processo quando tutti i campioni video saranno stati classificati con un t_i, sarà possibile definire i valori soglia. Tali valori permetteranno di distinguere la classe di appartenenza di ogni campione. Un approfondimento più esaustivo del t_i verrà sviluppato successivamente.

Mascheratura del background

Inizialmente per il processo di analisi era stato implementato un sistema che sfruttasse la libreria *OpenCV*, che permettesse di identificare gli oggetti in movimento all'interno dei video. Tale sistema permetteva tramite l'utilizzo della funzione di libreria *cv2.createBackgroundSubtractorKNN* di creare una mascheratura che andasse a selezionare solamente gli oggetti in movimento nella scena.

Una volta effettuata la mascheratura di tutto ciò che non fosse in movimento, veniva processato il frame del video in modo tale da selezionare solamente gli oggetti in movimento. Venivano poi applicati dei filtri di sfocatura gaussiana tramite la funzione *cv2.GaussianBlur* e filtri per la dilatazione tramite la funzione *cv2.dilate* in modo da rendere i bordi da rilevare più netti possibili. (nell'immagine 2.1 è possibile vedere il frame prima e dopo l'applicazione dei filtri. Prima - immagine in basso a sinistra, Dopo - immagine in basso a destra)

A questo punto, tramite la funzione `cv2.findContours` venivano rilevati i contorni dell'oggetto, incapsulandolo in un rettangolo. Di tale rettangolo veniva poi calcolato il punto centrale che rappresentava il centro di una ipotetica auto che si muoveva nell'inquadratura. Uno schema riassuntivo del funzionamento di questo metodo è riportato al codice 2.2

Codice 2.2: Schema di funzionamento del metodo ‘Mascheratura del background’.

```

...
subtractor = cv2.createBackgroundSubtractorKNN(...)
#funzione alternativa:
#subtractor = cv2.createBackgroundSubtractorMOG2(...)

...
fgMask = subtractor.apply(frame)
...

conts, _ = cv2.findContours(...)

...
for c in conts:
    x,y,w,h = cv2.boundingRect(c)
    xMid = int((x + (x+w))/2)
    yMid = int((y + (y+h))/2)
...

```

Tale sistema presentava diverse problematiche, una delle quali era la calibrazione necessaria per un corretto rilevamento degli oggetti. Infatti il sistema era molto sensibile alla variazione di luminosità della scena, rischiando così di non identificare le auto nelle ore di poca luce o di unire due auto se troppo ravvicinate. Il t.i in questo caso veniva calcolato come il numero di auto totali che in un campione video oltrepassavano un linea precedentemente definita. Tale criterio però non permetteva di distinguere la situazione in cui tutte le auto nell'inquadratura fossero state ferme, dalla situazione in cui non ci fossero auto nell'inquadratura, sia perché il sistema non avrebbe potuto identificare un'auto ferma, sia perché anche nel caso fosse stato possibile identificarla, non avrebbe oltrepassato la linea per incrementare il contatore.

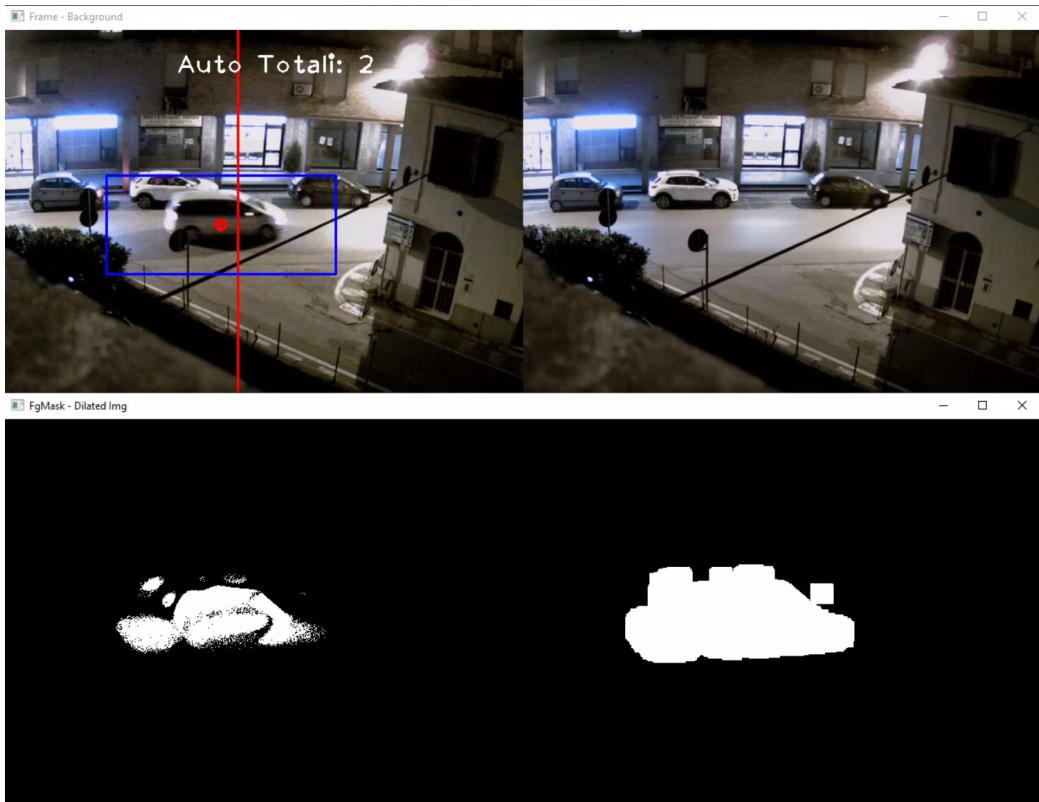


Figura 2.1: Esempio classificazione campione video con il primo sistema.

YOLO v5

Per i suddetti motivi è stato implementato un secondo sistema di rilevazione. Questo nuovo sistema permette di discriminare la tipologia di veicolo che viene identificata, oltre che tenere conto del tempo che impiegano i veicoli a transitare nell'inquadratura. Verrà quindi assegnato un t_i più elevato ad una singola auto che transita lentamente nell'inquadratura rispetto ad una stessa auto che transita a velocità più elevata.

Questa scelta è stata fatta per voler privilegiare un'analisi del traffico inteso come permanenza e numero di veicoli che percorrono una determinata strada e non come tempo per percorrere un tratto di strada da un punto ad un altro. Tutto ciò in considerazione di uno dei possibili ampliamenti futuri di questo progetto di tesi cioè l'analisi della qualità dell'aria basata su questo stesso dispositivo.

Questo nuovo sistema si basa su Yolov5, un algoritmo opensource che

permette il riconoscimento di oggetti basato sul COCO dataset¹. Il sistema è così strutturato: i campioni video vengono analizzati per frame, ogni frame viene processato tramite Yolov5, il quale permette di avere in uscita un elenco di tutti gli oggetti che vengono riconosciuti. In particolare le categorie utili alla nostra analisi sono Auto, Bus, Motoveicolo, Camion, Bicicletta.

Questo nuovo sistema è molto meno influenzato dalle condizioni di luce e risulta più affidabile e preciso. La categorizzazione dei veicoli presenti sulla scena permette di risolvere anche il secondo problema riscontrato nel primo metodo di analisi e cioè l'impossibilità di riconoscere veicoli fermi. Difatti questo nuovo sistema permette di riconoscere tutte le tipologie di veicoli anche se non in movimento o se molto ravvicinati fra loro.

Analisi con Yolov5

Verranno adesso illustrati degli esempi pratici nei quali è stato adoperato questo nuovo sistema di riconoscimento. Il primo esempio (Figura 2.2) permette di vedere come questo sistema si comporta con un traffico molto intenso di un'autostrada a 6 corsie, riuscendo a riconoscere molti veicoli presenti nell'inquadratura.

Un altro esempio (Figura 2.3) permette di vedere come il sistema riesca a identificare i veicoli anche se non pienamente visibili nell'inquadratura, difatti il sistema è riuscito a riconoscere il bus (contornato di rosso) pur essendo visibile solo la parte posteriore, e l'automobile all'estrema sinistra pur essendo visibile solo parte del cofano.

Un altro esempio (Figura 2.4) permette di notare come il sistema riconosca anche le persone, in questo caso i conducenti dei due motorini sulla sinistra. Tale identificazione però non apporterà nessun aumento del t_i del campione, in quanto solo i veicoli precedentemente menzionati apporteranno un incremento del t_i .

È stato quindi realizzato uno script Python “count_car_yolo_02.py” [13] che implementasse Yolov5 per il riconoscimento dei veicoli ed il calcolo del t_i del campione in analisi.

¹COCO: Common Objects in Context è un dataset contenente oltre 328000 immagini di oggetti comuni e persone, usato per il training di modelli per il riconoscimento automatico degli oggetti.

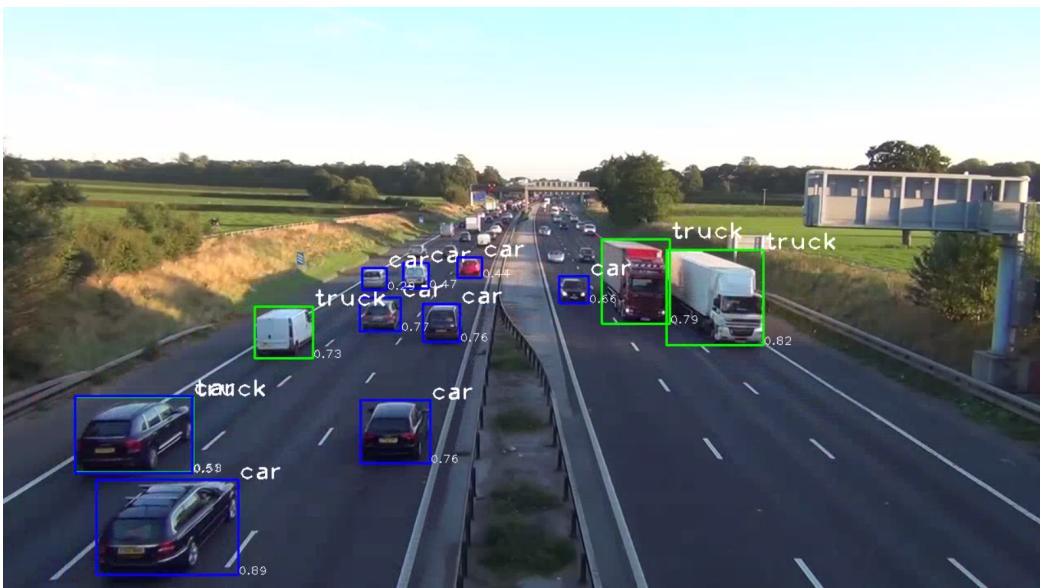


Figura 2.2: Sistema Yolo, Analisi con elevato traffico in un'autostrada a 6 corsie.

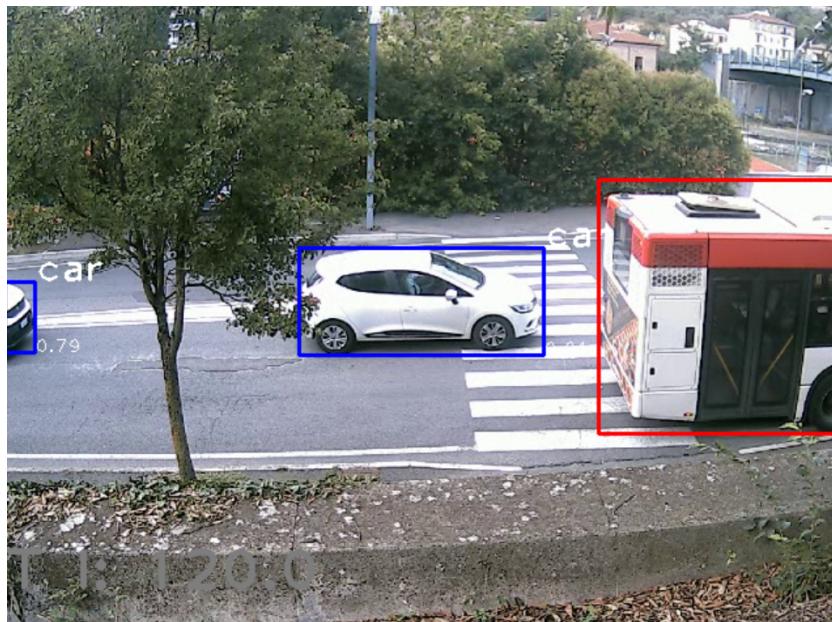


Figura 2.3: Sistema Yolo, caso particolare.

Tale sistema ha anche dei lati negativi, come la potenza computazionale necessaria per l'analisi dei campioni, a differenza del primo metodo che richiedeva una potenza computazionale molto inferiore. È stato possibile implementare in modo efficiente il riconoscimento dei veicoli tramite Yolov5

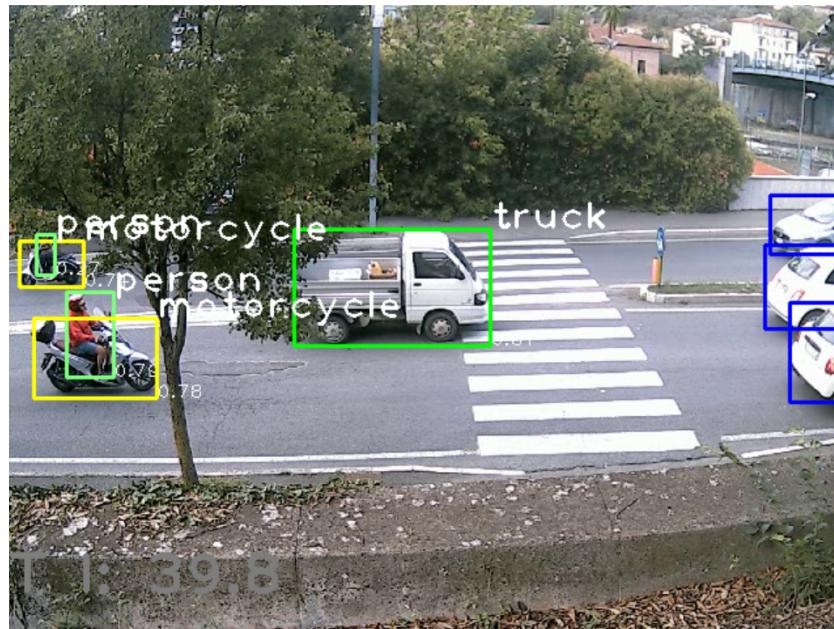


Figura 2.4: Sistema Yolo, caso particolare.

in quanto per l’analisi è stato utilizzato un computer² dotato di una scheda video con architettura CUDA. Sfruttando così la potenza computazionale della scheda video, il tempo necessario per l’analisi di ogni singolo frame si è ridotto drasticamente, passando da 8 s per l’analisi di un campione di 15 s a 1.2 s per l’analisi dello stesso campione. Per entrambe le analisi è stato usato il medesimo computer, sfruttando nel primo caso la potenza computazionale del processore e nel secondo caso della scheda video, ottenendo così un decremento di oltre 6 volte del tempo necessario all’analisi.

count_car_yolo_02.py

Sicuramente uno degli script più importanti per la realizzazione di questo progetto di tesi è proprio “count_car_yolo_02.py” [13]. Tale script permette sia il riconoscimento dei veicoli tramite il sistema Yolov5, sia il calcolo del valore t_i per ogni campione registrato. Inizialmente vengono definiti tutti i parametri necessari ad un corretto funzionamento dello script, successivamente tramite la libreria *OpenCV* viene gestito il flusso video dei campioni registrati. Tramite la funzione *cv2.fillPoly* è possibile andare ad oscurare

²Specifiche computer: Nvidea RTX 3060 (mobile), Ryzen 5 5600H, 16 gb RAM.

aree dei campioni, ad esempio oscurare auto parcheggiate, se presenti nell'inquadratura, che andrebbero a falsare il conto del t.i del campione. La funzione *cv2.fillPoly* funziona inserendo le coordinate dei vertici di un poligono, il quale verrà riempito al suo interno per oscurare la scena sottostante. Per facilitare questa procedura è stata realizzata una funzione apposita nello script “*main_analisi.py*”, più precisamente la funzione *draw_black_box()*. Tale funzione permette di aprire un file video e tramite il puntatore segnare i 4 estremi del poligono che vogliamo tracciare. Verrà restituito nel terminale un vettore contenente le coordinate selezionate, in modo tale da poterle inserire nello script “*count_car_yolo_02.py*” per effettuare il mascheramento.

Una volta completata la fase di mascheramento delle zone desiderate, il frame viene processato con l'ausilio di Yolov5. Per questo progetto di tesi è stato utilizzata la versione Yolov5s, in quanto permetteva di avere un ottimo livello di riconoscimento, assieme ad elevate prestazioni.

Codice 2.3: Schema di funzionamento del metodo 'Yolo'.

```
...
#main_analisi.py
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
#count_car_yolo_02.py
results = model(frame)
...
df = results.pandas().xyxy[0]
car_found = (df.name.values == "car").sum()
bus_found = (df.name.values == "bus").sum()
motorbike_found = (df.name.values == "motorbike").sum()
truck_found = (df.name.values == "truck").sum()
bicycle_found = (df.name.values == "bicycle").sum()
```

Nello script “*main_analisi.py*” viene caricato il modello yolov5s tramite la libreria *torch*, quindi viene passato come argomento della funzione *count_veichle()* nello script “*count_car_yolo_2.py*”. Questo permette di non dover caricare il modello Yolov5s ogni volta che viene analizzato un frame, visto che tale operazione richiede del tempo aggiuntivo. Una volta che il frame è stato analizzato vengono estratte le categorie di interesse [Vedi tabella

2.1]. Tramite questo script è possibile anche abilitare una visualizzazione in tempo reale delle analisi: se il parametro *show_bool* diventa *True* viene visualizzato il frame, con evidenziati i veicoli riconosciuti ed un contatore del *t_i* che varia nel tempo.

Traffic index

Per il calcolo del *t_i*, invece di contare le auto che attraversano una predefinita linea, è stato deciso di adottare un approccio differente. Difatti il precedente sistema presentava delle problematiche come l'impossibilità di distinguere la situazione nella quale tutte le auto fossero ferme dalla situazione in cui non ci fossero auto presenti nella scena. Per risolvere questo problema è stato adottato il seguente metodo: per ogni frame analizzato vengono identificate varie tipologie di veicolo (Auto, Bus, Motoveicolo, Camion, Bicicletta) ad ogni veicolo viene associato un valore PCU (Passenger Car Unit). Tale valore permette di formalizzare il concetto per il quale un camion produrrà più traffico rispetto ad una bicicletta, definendo infatti una metrica di traffico generato rispetto al traffico prodotto da un'auto passeggeri, tenendo in considerazione parametri come dimensioni, velocità e densità. Ad ogni tipologia di veicolo viene quindi associato un valore PCU, successivamente

Tabella 2.1: Categorie e PCU associati.

Categoria	PCU [14]
Auto	1
Autobus	2
Motoveicolo	0.4
Camion	1.9
Bicicletta	0.2

alla categorizzazione di ogni frame vengono moltiplicati il valore PCU correlato per ogni veicolo con il numero di veicoli identificati, per ottenere così un risultato parziale di *t_i*. Tale procedura viene effettuata per ogni frame del campione video, alla fine di questo processo otteniamo così un valore di *t_i*, somma di tutti i *t_i* dei ogni frame, che sia rappresentativo di tutti i veicoli identificati. In questo modo è stato possibile creare un metodo per il calco-

lo del t_i che possa differenziare la presenza di veicoli fermi dall'assenza di veicoli, permettendo anche di differenziare un'auto che passa velocemente da una che passa lentamente, associando un valore più alto di t_i a quest'ultima in quanto sarà presente in più frame rispetto ad un'auto più veloce.

Una volta completata la fase di analisi e calcolo del t_i, viene generato un file contenente tutti i nomi dei campioni analizzati con associati anche i rispettivi valori del t_i. Questo permette di salvare lo stato dell'analisi così da non doverla ripetere in caso di problematiche, oltre a dare il tempo necessario per la valutazione dei livelli soglia per l'assegnazione delle classi.

2.2.2 Scelta valori soglia

Prima di effettuare l'esportazione dei campioni è necessario scegliere i valori soglia per cui un determinato campione rientri in una delle possibili classi. Per facilitare questa fase sono stati sviluppati vari strumenti nello script “main_analisi.py”. Il primo strumento permette di visualizzare sotto forma di grafico il file generato dalla fase di analisi tramite la funzione *print_graph_from_file()*. Tale funzione sfruttando la libreria *matplotlib* permette di visualizzare sull'asse delle ascisse il numero del campione (utile per lo strumento successivo) mentre sull'asse delle ordinate il valore t_i assegnato a tale campione. In questo modo è possibile visualizzare un andamento dei campioni nel tempo, visionando quindi i livelli massimi e minimi raggiunti.

Un secondo strumento sviluppato appositamente per questa fase permette la visualizzazione del campione video correlato in base al valore delle ascisse del grafico precedente. Difatti la funzione *Play_video()* permette, selezionando il file utilizzato per la generazione del grafico, di andare a selezionare il file video correlato ad ogni campione e avviandone la riproduzione. In questo modo, è possibile avere un feedback visivo del livello di traffico di ogni specifico campione.

Alla fine di questa fase di analisi dovranno, per questo progetto, essere scelti due livelli soglia, il primo che delimiti i campioni con traffico *Assente* da i campioni con traffico *Presente*, e il secondo che definisca il limite tra campioni con traffico *Presente* dai campioni con traffico *Intenso*. Tali scelte sono strettamente dipendenti dalla tipologia di strada in analisi e dalla con-

formazione della stessa. Una volta definiti questi livelli soglia sarà possibile procedere con l'ultima fase di analisi: l'esportazione dei campioni audio.

2.2.3 Export campioni audio

L'ultima fase prima del caricamento dei campioni audio su EdgeImpulse è la loro esportazione. Per eseguire questa funzione è stata sviluppata una funzione apposita `export_audio_file()` nello script “`main_analisi.py`”. Tale funzione permette di rinominare i file audio, con anteposto il nome della classe, nella forma “`NomeClasse.T_i.Timestamp.json`”; questa forma permette a EdgeImpulse di riconoscere la classe automaticamente, permettendo inoltre di mantenere le informazioni del `t_i` e della data di acquisizione del campione.

Il funzionamento di questa funzione è semplice: inizialmente deve essere selezionato il file che viene generato con la funzione `video_analyze()`, nel quale sono salvati i nomi di tutti i campioni con associato il proprio valore di `t_i`. Successivamente vengono richiesti i valori soglia delle classi: in questo caso essendo due classi devono essere specificati 2 valori. La funzione itera su tutta la lista dei campioni rinominando i file con la classe associata nella forma detta precedentemente. Contemporaneamente viene generato un nuovo file “`export_audio_log.txt`” nel quale sono salvati i nomi dei campioni, il livello `t_i` associato e anche la classe nella quale il campione è stato inserito.

La creazione di questo file è utile per effettuare un controllo aggiuntivo del dataset, infatti tramite la funzione `check_export()` è possibile selezionare il file appena generato e chiedere tramite l'interfaccia di visionare tutti i campioni appartenenti ad una specifica classe. Sono riprodotti quindi tutti i campioni video appartenenti alla classe selezionata: al termine di ogni riproduzione video è possibile assegnare al campione una classe differente dalla classe iniziale, oppure lasciarlo nella sua classe attuale passando al campione successivo. Nel caso di una modifica della classe viene fatto un backup del file e viene generato un nuovo file “`export_audio_log.txt`” con la classe modificata. Sarà possibile poi convalidare tutte le modifiche tramite la funzione `change_validation()`, la quale permette di rinominare i file audio in modo tale che le modifiche effettuate con la funzione `check_export()` siano applicate.

Capitolo 3

EdgeImpulse

Una volta completato il dataset, è necessario caricare i dati acquisiti su EdgeImpulse per la creazione del modello di rete neurale che permetterà il riconoscimento del livello di traffico. Per svolgere questa operazione EdgeImpulse mette a disposizione una procedura che guida l'utente in tutte le fasi fondamentali di tale operazione, dalla creazione di un *Impulse*¹ all'addestramento della rete neurale fino all'esportazione del progetto sotto forma ad esempio di libreria Arduino.

3.1 EdgeImpulse

La prima fase da svolgere è il caricamento del dataset correttamente suddiviso nelle classi che vogliamo andare ad analizzare. Per fare ciò basterà recarsi nella sezione “Data Acquisition” per poi effettuare il caricamento dei dati. Dopo aver selezionato tutti i campioni che vogliamo caricare sulla piattaforma, EdgeImpulse ci propone una feature che permette di suddividere il materiale caricato in modo automatico nel test set² e nel training set³, con una ripartizione 80% training, 20% test. Prima del caricamento è possibile abilitare l'opzione “Infer from filename”, ciò permette l'assegnazione auto-

¹Schema generale di analisi composto principalmente da 3 blocchi: input block, processing block e learning block.

²Set di dati usato per la validazione di un corretto addestramento della rete neurale.

³Set di dati usato per l'addestramento della rete neurale.

matica della classe al campione in base al nome del campione stesso seguendo la forma “NomeClasse.(altro).estensione”.

Completata la fase di caricamento dei dati il sistema richiede la creazione di un *Impulse* e quindi dello schema che dovrà essere seguito per l’addestramento della rete neurale.

Il primo blocco “input block” permette di definire i parametri di ingresso dei file audio quali *Window size*⁴ e *Window increase*⁵.

Il secondo blocco “processing block” definisce i processi che i file dovranno subire prima di passare all’ultimo blocco. Tale sezione è di fondamentale importanza per ottenere dei buoni risultati, in quanto non evidenziare le corrette caratteristiche di un campione potrebbe non permettere al sistema di identificarli correttamente. Ci sono vari preset, con associata una descrizione tra cui è possibile scegliere: nel caso in questione è stato scelto di utilizzare *Audio (MFE)* in quanto era il preset che dava le migliori prestazioni.

Infine deve essere creato il “learning block”, anche in questo caso ci sono vari preset tra cui è possibile scegliere, in base alle proprie esigenze, nel caso di questo progetto è stato deciso di utilizzare *Classification (Keras)*.

Una volta completato il processo di creazione dell’*Impulse*, il sistema permette di definire i parametri del “processing block” e successivamente del “learning block”. Dopo aver configurato i due blocchi è possibile iniziare l’addestramento. Al termine di questa fase verrà restituito un valore di accuratezza sul validation set, la confusion matrix che permette di visualizzare le classi nelle quale il sistema performa in modo migliore e le classi in cui i risultati non sono ottimali, oltre che una stima delle prestazioni che il modello appena generato avrà sul dispositivo finale.

Per lo sviluppo di questo progetto di tesi sono stati testate varie configurazioni fino a trovare quella che permetteva di avere dei risultati ottimali in base allo scopo applicativo. I singoli parametri utilizzati saranno descritti successivamente.

⁴Lunghezza in millisecondi dei campioni usati durante l’addestramento.

⁵Parametro che permette di definire la frequenza dei campioni, variando tale parametro è possibile effettuare un sovraccampionamento o sottocampionamento.

3.2 Caso di studio

In questo progetto sono stati analizzati principalmente due spot di rilevazione (Figura 3.1), il primo in una strada a senso unico ad unica corsia in una città di provincia (Colle di Val d’Elsa), il secondo all’ingresso di una rotatoria nella città di Siena.

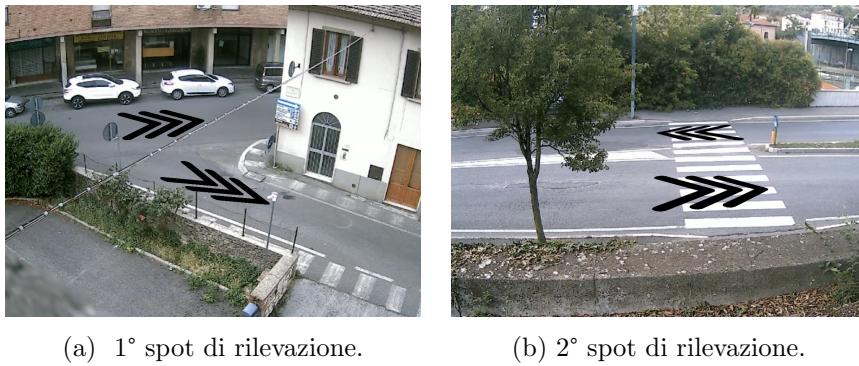


Figura 3.1: Spot di rilevazione con indicati i sensi di percorrenza dei veicoli.

3.2.1 1° Spot di rilevazione

Nel primo spot di rilevazione il traffico non è molto intenso durante tutta la giornata: pur avendo acquisito campioni per un totale di 63 ore (in vari momenti della giornata) non è stato possibile avere una quantità sufficiente di campioni con traffico che potesse appartenere alla categoria *Intenso*, in quanto troppo simili alla categoria *Presente*. Questa caratteristica ha causato dei problemi durante il processo di addestramento, in quanto non c’era una distinzione netta tra il traffico nella classe intermedia *Presente* e quello della classe di traffico elevato *Intenso*.

Questa caratteristica può essere notata dalla confusion matrix (Figura 3.2) restituita da EdgeImpulse dopo l’addestramento della rete neurale. Come è possibile vedere, l’accuratezza nell’identificare la classe *Assente* è abbastanza buona, invece il sistema tende a non distinguere correttamente le classi *Inteso* e *Presente*.

Per questa caratteristica intrinseca della strada è stato deciso di testare il sistema anche in un secondo spot di rilevazione, uno spot nel quale si

presentasse più frequentemente la condizione di traffico intenso, avendo così una distinzione più netta nei campioni delle tre classi da identificare.

Per fare ciò è stato identificato un nuovo spot di rilevazione che conciliasse un elevato flusso di auto con la possibilità di effettuare le rilevazioni in modo semplice.

	ASSENTE	INTENSO	PRESENTA
ASSENTE	88.9%	5.6%	5.6%
INTENSO	6.1%	68.9%	25%
PRESENTA	8.7%	44.4%	46.8%
F1 SCORE	0.88	0.65	0.51

Figura 3.2: Confusion matrix 1° spot di rilevazione.

3.2.2 2° Spot di rilevazione

Il secondo spot di rilevazione, anche grazie alla sua posizione, tende ad avere momenti di traffico più intensi durante la giornata, soprattutto nelle ore di punta. Questa caratteristica ha apportato un notevole miglioramento nella capacità di classificazione delle condizioni stradali, in quanto i campioni con traffico *Intenso* sono nettamente diversi dai campioni con traffico *Presente*.

In questo nuovo spot di rilevazione sono stati acquisiti un totale di 6 ore di dati. Alla fine del processo di campionamento tutti i dati sono stati analizzati ed è stato estrapolato un totale di 20 minuti di traffico *Intenso*. Conseguentemente sono stati presi in modo casuale circa 20 minuti delle altre due classi, per raggiungere un totale di 660 campioni per circa 1 ora e 10 minuti totali.

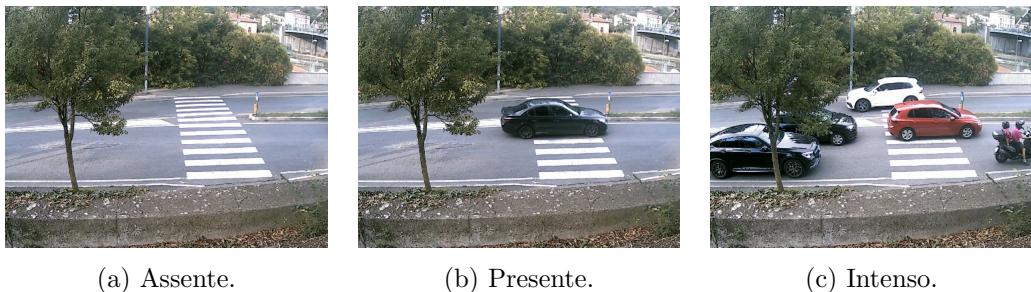


Figura 3.3: Vari livelli di traffico nel 2° spot.

Una volta selezionati i dati rilevanti è stato creato un nuovo progetto su EdgeImpulse, quindi sono stati caricati tutti i dati ed è stato creato un nuovo *Impulse*.

Dopo vari test e classificazioni in cui sono stati variati i vari parametri è stata trovata una combinazione di impostazioni che permette di ottenere ottimi risultati in termini di accuratezza (Tabella 3.1).

Input block	Processing block	Learning block
Windows size: 4000 ms Windows increase: 2000 ms Frequency: 16000 Hz	Audio (MFE)	Classification (Keras) Output features: 3

Tabella 3.1: Parametri impulse.

In particolare nel Processing block sono stati impostati i parametri riportati nella tabella 3.2.

Impostazione	Valore
Frame length	0.02
Frame stride	0.01
Filter number	40
FFT length	256
Low frequency	250
High frequency	0
Noise floor (dB)	-40

Tabella 3.2: Parametri Processing block.

Per quanto riguarda il Learning block i parametri specifici sono riportati nella tabella 3.3.

Impostazione	Valore
Number of training cycles	100
Learning rate	0.0045
Validation set size	20 %
Auto-balance dataset	Not Set
Data augmentation	Not Set

Tabella 3.3: Parametri Learning block.

La struttura della rete neurale è composta come riportato in tabella 3.4.

Con le precedenti impostazioni è stato possibile raggiungere un'accuratezza del 92.8% sul validation set (Figura 3.4) ed un accuratezza del 91.52% sul test set (Figura 3.5).

Come è possibile notare, i risultati ottenuti sono ottimi per gli scopi di questo progetto di tesi, con una leggera diminuzione dell'accuratezza nel Test set ma di poca rilevanza.

Struttura rete neurale
Input layer (15,960 features)
Reshape layer (40 columns)
1D conv / pool layer (8 neurons, 3 kernel size, 1 layer)
Dropout (rate 0.25)
1D conv / pool layer (16 neurons, 3 kernel size, 1 layer)
Dropout (rate 0.25)
Flatten layer
Output layer (3 classes)

Tabella 3.4: Parametri Rete neurale.

Last training performance (validation set)



Confusion matrix (validation set)

	ASSENTE	INTENSO	PRESENTA
ASSENTE	87.9%	5.2%	6.9%
INTENSO	8.6%	91.4%	0%
PRESENTA	3.4%	0%	96.6%
F1 SCORE	0.89	0.91	0.96

Figura 3.4: Accuratezza e Confusion matrix sul Validation set.



	ASSENTE	INTENSO	PRESENTA	UNCERTAIN
ASSENTE	85.5%	6.5%	8.1%	0%
INTENSO	4.2%	95.8%	0%	0%
PRESENTA	7.0%	0%	93.0%	0%
F1 SCORE	0.85	0.94	0.94	

Figura 3.5: Accuratezza e Confusion matrix sul Test set.

3.3 Esportazione del Modello

Una volta completato il processo di addestramento tramite la piattaforma EdgeImpulse, è possibile andare a esportare una libreria Arduino che implementa nativamente il modello necessario per il funzionamento della rete neurale per la predizione della classe di appartenenza di un campione audio.

Per svolgere questa operazione è sufficiente recarsi nel menù “Deployment” del progetto di EdgeImpulse e selezionare “Arduino Library”.

Opzionalmente è possibile anche abilitare l’ottimizzazione del modello tramite il compilatore EON™ che permette di ridurre fino al 50% le richieste di memoria a parità delle medesime prestazioni. Viene visualizzata anche una stima delle prestazioni con e senza questa ottimizzazione mostrando: una stima di utilizzo della RAM, latenza, accuratezza e memoria occupata.

A questo punto sarà necessario solamente premere il pulsante “Build” per la creazione della libreria Arduino, la quale potrà essere caricata come una libreria esterna sull’Arduino IDE. Dopo aver aggiunto correttamente il file .zip appena generato, tra gli esempi di libreria saranno presenti degli esempi che permettono di testare facilmente il modello generato.

Per questo progetto di tesi l’esempio di libreria di riferimento è “nano_ble33_sense_microphone”. Tale esempio permette di effettuare analisi audio alla fine delle quali vengono restituiti tramite porta seriale le percentuali di appartenenza a tutte e tre le classi possibili.

Per testare il sistema appena realizzato è stata creata una funzione appropriata *VideoTesting()* nello script “main_analisi.py” tramite il quale è possibile visualizzare e salvare localmente un video generato da una webcam con in sovrapposizione i dati trasmessi dall’Arduino con evidenziate le classi rilevate.

Conclusioni

Alla fine di questo progetto di tesi è stato realizzato un dispositivo in grado di rilevare con una più che buona accuratezza il livello di traffico del secondo spot di rilevazione. Sicuramente il luogo dove questo progetto viene applicato ricopre un aspetto molto importante per una buona riuscita dello stesso. Un luogo nel quale il traffico non è troppo presente potrebbe rendere molto difficile la fase di training del dispositivo, non avendo a sufficienza materiale per l'addestramento. Sicuramente la fase di acquisizione dei dati è anch'essa cruciale per la buona riuscita del progetto. Infatti, se i campioni non sono equamente suddivisi nelle varie classi da identificare, potrebbero sorgere anche in questo caso problemi in fase di addestramento della rete neurale.

Il sistema come è stato studiato in questo progetto è in grado di rilevare in modo autonomo il livello di traffico. Attualmente il sistema deve essere collegato via cavo con un computer per leggerne le uscite. Sviluppi futuri potrebbero riguardare la resa indipendente del dispositivo, dotandolo di una rete di comunicazione e facendo in modo, ad esempio, che trasmetta l'informazione del traffico solamente nel momento in cui viene rilevato il livello più alto di traffico *Intenso*, oppure che possa operare con altre modalità a seconda della specifica applicazione finale.

Come accennato inizialmente, una delle possibili future applicazioni di questo progetto risiede proprio nella possibilità di creare un dispositivo che prenda questo come modello base per la realizzazione di un sistema per la rilevazione della qualità dell'aria come la stima della percentuale di CO₂ o la stima del livello di polvere sottili, il tutto legato alla presenza di vari livelli di traffico, così da verificare se esista, e come possa variare, una correlazione diretta tra i vari livelli di traffico e i livelli di inquinanti nell'aria.

Un altro aspetto da analizzare potrebbe essere quello del livello di generalizzazione raggiunto in questo progetto. Il sistema riesce a raggiungere un ottimo livello di accuratezza con campioni analizzati nel medesimo luogo nel quale è stato effettuato il training. Un possibile sviluppo futuro potrebbe risiedere nell'analizzare in che modo questo sistema possa essere utilizzabile in diversi luoghi pur non effettuando nuovamente l'addestramento della rete neurale.

Bibliografia

- [1] “Arduino nano 33 ble sense Datasheet.” [Online]. Available: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>
- [2] “Dispositivi Embedded supportati dalla piattaforma EdgeImpulse.” [Online]. Available: <https://docs.edgeimpulse.com/docs/development-platforms/fully-supported-development-boards>
- [3] “Microcontrollore,” page Version ID: 124768100. [Online]. Available: <https://it.wikipedia.org/w/index.php?title=Microcontrollore&oldid=124768100>
- [4] L. Dutta and S. Bharali, “Tinyml meets iot: A comprehensive survey,” *Internet of Things*, 2021.
- [5] “Datasheet microfono MP34DT05.” [Online]. Available: https://content.arduino.cc/assets/Nano_BLE_Sense_mp34dt05-a.pdf
- [6] S. H. Hong, S. J. Lee, and H. J. Park, “Cos mems system design with embedded technology,” *KEPCO Journal on Electric Power and Energy*, 2020.
- [7] Thomas Kite, Ph.D. VP Engineering Audio Precision, Inc., “Understanding PDM Digital Audio,” 2012. [Online]. Available: https://users.ece.utexas.edu/~bevans/courses/rtdsp/lectures/10_Data_Conversion/AP_Understanding_PDM_Digital_Audio.pdf
- [8] F. Maccantelli, “Github / PDM_sender_1.0.ino,” 2022. [Online]. Available: https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/PDM_SENDER_1.0/PDM_SENDER_1.0.ino

- [9] F. Maccantelli, “Github / record_data.py,” 2022. [Online]. Available: [https://github.com/macca0612/](https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/record_data.py)
- [10] F. Maccantelli, “Github / rec_audio.py,” 2022. [Online]. Available: [https://github.com/macca0612/](https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/rec_audio.py)
- [11] F. Maccantelli, “Github / rec_video.py,” 2022. [Online]. Available: [https://github.com/macca0612/](https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/rec_video.py)
- [12] F. Maccantelli, “Github / main_analisi.py,” 2022. [Online]. Available: [https://github.com/macca0612/](https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/main_analisi.py)
- [13] F. Maccantelli, “Github / count_car_yolo_02.py,” 2022. [Online]. Available: [https://github.com/macca0612/](https://github.com/macca0612/Audio-based-classification-with-TinyML-on-embeddeddevice/blob/main/count_car_yolo_02.py)
- [14] T. for London, “Traffic modelling guidelines v4,” 2021. [Online]. Available: <https://content.tfl.gov.uk/traffic-modelling-guidelines.pdf>