UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sviluppo e valutazione di un sistema di rilevamento automatico della violenza all'interno di video

Development and evaluation of a system for the automatic violence detection in videos

Relatore:
Prof. Aldo Franco Dragoni

Tesi di Laurea di:
Simone Cappanera

Correlatore:
Dott. Paolo Sernani

A.A. 2019/2020

Ringraziamenti

Grazie ad Alessia per essere stata sempre al mio fianco a supportarmi ed incoraggiarmi durante la scrittura di questo elaborato, ricordandomi ogni volta quanto io sia speciale.

Grazie alla mia famiglia per avermi accompagnato durante questa prima parte del percorso universitario, sostenendo tutte le mie scelte e permettendomi di arrivare a questo traguardo.

Grazie al professore Aldo Franco Dragoni che è sempre stato un grande modello a cui ispirarsi e che fece nascere in me durante il primo anno di università la grande passione che nutro per il magnifico ambito dell'Intelligenza Artificiale.

Grazie infinite al mio correlatore Paolo Sernani per la sua grande disponibilità e per avermi aiutato durante il mio periodo di tirocinio.

Grazie ai miei compagni di corso Lisa, Michelangelo, Matteo e Simone per essermi stati vicino durante tutte le sfide affrontate in questi primi tre anni del mio percorso universitario e per tutti i momenti felici e divertenti passati insieme.

Un enorme grazie, infine, a Simone Accattoli, il cui lavoro di tesi di Laurea Magistrale è stato per me una grandissima fonte d'ispirazione da cui partire e prendere spunto per il mio tirocinio e il mio elaborato finale.

Ancona, Dicembre 2020

Cappanera Simone

Abstract

La violenza è sempre più diffusa nella nostra società e si è alla continua ricerca di nuove soluzioni per combatterla. La videosorveglianza è sempre stata uno degli strumenti più utili a tale scopo permettendo di rafforzare la sicurezza sia in ambienti pubblici che privati. Nonostante però i suoi numerosi vantaggi, gli odierni sistemi di sorveglianza risultano poco efficaci in molte situazioni. Questi, infatti, sono solitamente monitorati da operatori umani e quindi non si ha la certezza che l'operatore sia attento e vigile in ogni momento a ciò che sta succedendo nelle scene riprese. Questo fa sì che spesso i sistemi di videosorveglianza si riducano ad un semplice strumento per le indagini a posteriori di eventi spiacevoli. Un sistema automatico in grado di eseguire il monitoraggio continuo dei video porterebbe a migliorare decisamente l'efficienza di questi sistemi. In questo elaborato viene riproposta una soluzione già adottata in un lavoro di tesi precedente di uno studente dell'Università Politecnica delle Marche, Simone Accattoli, basata sulle 3D Convolutional Neural Network [1], in grado di rilevare lotte, movimenti aggressivi e scene di violenza nei video, effettuando però un porting dell'architettura realizzata da Accattoli utilizzando le più recenti tecnologie. Questo metodo aveva già mostrato nel lavoro precedente di avere performance molto promettenti, anche rispetto alle tecniche finora definite nello stato dell'arte, su tre dataset di benchmark usati in fase di test (Hockey Fights dataset, Crowd Violence dataset e Movie Violence dataset) e in questa tesi si ha avuta la conferma di quei risultati.

INDICE

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Con il termine “violenza” si intende, in generale, un atto volontario, esercitato da un soggetto su un altro, in modo da determinarlo ad agire contro la sua volontà. Infatti, etimologicamente, violenza significa "che vìola", cioè ciò che oltrepassa il limite della volontà altrui. La violenza tra gli uomini è un'azione compiuta mediante l'abuso della forza di una o più persone che provoca dolore ad altri individui, anche indirettamente, danneggiandoli. La violenza, inoltre, non ha un’unica forma, ma ha diverse manifestazioni, infatti può essere sia fisica, con o senza armi, sia verbale fino ad arrivare a quella psicologica, quando ha ripercussioni sulla struttura psichica e i contenuti della coscienza. [2] Nella società odierna la violenza è sempre più diffusa, infatti i media ci riempiono in continuazione di notizie che raccontano eventi violenti ed aggressivi, basti pensare che ogni giorno sentiamo parlare di delitti, maltrattamenti, abusi, omicidi e vari tipi di violenze, da quelle di genere fino ad arrivare a quelle razziali. A questo punto è naturale chiedersi cosa ci sia all’origine di tutta questa violenza. Un famoso aforisma attribuito ad un’antica leggenda Cherokee recita: “*Ci sono due lupi in ognuno di noi. Uno è cattivo e vive di odio, gelosia, invidia, menzogna ed egoismo. L’altro è il lupo buono. Vive di pace, amore, speranza, generosità, compassione, umiltà e fede. E quale lupo vince? Quello che nutri di più.”*. Quindi, secondo questa massima, l’animo umano è fatto di impulsi positivi e negativi e sta a ognuno di noi decidere quale di questi far prevalere. Inoltre, possiamo affermare che l’origine della violenza, quindi, non è soltanto insita nella natura umana, ma è anche e soprattutto frutto di diverse componenti istintive negative che ognuno di noi ha e che abbiamo il dovere di tenere a freno grazie alla razionalità e il buonsenso. Il lavoro di tesi qui svolto si basa, come quello di Accattoli [1], sull’interpretazione della violenza come una azione volontaria attuata come costrizione materiale.

In questo contesto, la *Violence Detection* ha come obiettivo quello di riconoscere in maniera automatica un’aggressione tra due o più persone. L’architettura sviluppata non è altro che un porting utilizzando tecnologie più attuali del sistema che era stato sviluppato nella Tesi citata precedentemente [1] che utilizzava una rete neurale convoluzionale 3D per rilevare il verificarsi di un atto violento all’interno di video, che potrebbero essere verosimilmente quelli ripresi da un sistema di videosorveglianza.

1.1 – Motivazioni

Per combattere il fenomeno della violenza e assicurare così la protezione delle persone da atti violenti, i sistemi di videosorveglianza sono tra gli strumenti più utili, visto che permettono di monitorare potenziali comportamenti anomali ed indesiderati. Il problema, però, sta nel fatto che solitamente questo monitoraggio viene eseguito da un operatore umano che, a differenza di una macchina, può distrarsi, addormentarsi o anche semplicemente non accorgersi del verificarsi di un determinato evento a causa dei numerosi monitor da controllare. Questo, quindi, può portare ad un ritardo nella chiamata dei soccorsi che può anche essere questione di vita o di morte.

Il progetto originale sviluppato da Accattoli [1] nasceva dall'idea di poter creare un supporto ai sistemi di sorveglianza odierni per il riconoscimento automatico di scene di violenza, al fine di migliorare la sicurezza delle persone e fornire una immediata risposta in azione ad una violenza in atto. La sua Tesi partiva dalle tecniche utilizzate per la *human action recognition*, per poi passare all'analisi dello stato dell'arte della *violence detection*, infatti quest'ultima non è che un'evoluzione della prima consistendo nel riconoscimento di una particolare azione o comportamento che classifichiamo come violento. Accattoli aveva deciso a partire da questa analisi di utilizzare un modello di rete convoluzionale 3D pre-addestrato chiamato C3D [3] come descrittore del flusso video e di attuare la classificazione della presenza o meno di un'aggressione mediante l'uso di una *Support Vector Machine* (SVM), classificatore non lineare molto utilizzato in letteratura.

Questo lavoro, invece, nasce dall'idea di effettuare un porting dell'architettura realizzata da Accattoli utilizzando tecnologie più avanzate di quelle che erano state utilizzate nel suo progetto, ammodernando così la struttura complessiva del modello precedentemente descritto e ottimizzando il codice e le sue prestazioni.

I campi applicativi, come era stato sottolineato da Accattoli nel suo elaborato, sono numerosi e a titolo di esempio riportiamo qui di seguito quelli che da lui erano stati messi in luce [1]:

- *Videosorveglianza.* Ogni ambiente che necessiti di telecamere di videosorveglianza può far uso di questo sistema per migliorare la sicurezza.
- *Riconoscimento di contenuti online violenti.* Si cerca di limitare la fruizione online di video con contenuti violenti.
- *Segmentazione di filmati di lunga durata.* Poter estrarre da un filmato la porzione di frame in cui si presenta un'aggressione per aiutare così la costruzione di un dataset più significativo.

1.2 – Obiettivi

Gli obiettivi alla base di questo lavoro di tesi sono i seguenti:

- Sviluppare un sistema per il riconoscimento automatico delle scene di violenza tra due o più persone, basato su una rete convoluzionale 3D, che supporti i sistemi di sorveglianza nel rivelare crimini effettuando un porting dell'architettura realizzata da Accattoli utilizzando tecnologie più avanzate di quelle che erano state utilizzate nel suo progetto.
- Rimostrare nuovamente come una rete neurale profonda pre-addestrata su un dataset differente da quello su cui si vuole lavorare, possa essere comunque utilizzata come un estrattore di features.
- Ripetere gli stessi esperimenti realizzati nel lavoro di tesi di Accattoli utilizzando la nuova architettura prodotta con il porting e confrontare i risultati ottenuti con quelli della tesi precedente e con quelli degli approcci già presenti nello stato dell'arte sui tre datasets di benchmark che vengono principalmente utilizzati per la *violence detection*.

1.3 – Struttura della tesi

Il presente elaborato è suddiviso nel modo seguente:

- Nel secondo capitolo introdurremo velocemente gli aspetti principali dello stato dell'arte riguardante l'argomento trattato. In particolare, ci focalizzeremo sulle tecniche maggiormente utilizzate per quanto riguarda il *deep learning* e la sua applicazione al problema della *violence detection*.
- Nel terzo capitolo viene ripresa la teoria alla base dell'approccio selezionato e la struttura della rete C3D.
- Nel quarto capitolo andremo a illustrare l'intero sistema realizzato, indicando i vari strumenti utilizzati, i tre benchmark dataset, l'architettura del sistema di riconoscimento con le nuove tecnologie utilizzate e infine l'addestramento e il testing del classificatore.
- Nel quinto capitolo vengono mostrati i vari esperimenti eseguiti confrontando i risultati con quelli degli approcci dello stato dell'arte su tre famosi dataset di benchmark e inoltre verranno svolti degli esperimenti anche su un dataset realizzato dall'AIRTLab[1].
- Infine, il sesto ed ultimo capitolo mostra l'analisi conclusiva del progetto con i possibili sviluppi futuri.

[1] Artificial Intelligence and Real Time Systems Laboratory, Dipartimento di Ingegneria dell'informazione, Università Politecnica delle Marche.

Capitolo 2

Lo Stato dell'Arte

In questo capitolo verranno illustrati sinteticamente gli aspetti fondamentali della *violence detection* che, invece, erano stati approfonditi nel dettaglio nel lavoro di tesi di Accattoli [1], andando ad analizzare i concetti teorici e tecnologici presenti nello stato dell'arte, soffermandoci, però, in particolare, sulle tecniche basate sul *deep learning*. Inoltre, indicheremo le motivazioni alla base della scelta del metodo utilizzato in questo progetto, che ovviamente erano quelle già messe in luce da Accattoli [1] nel suo elaborato.

2.1 - Violence Detection

Come era stato già accennato nel capitolo introduttivo, la *violence detection* deriva dalla *human action recognition*. Come evidenziato da Accattoli nel suo lavoro di tesi [1], quest'ultima utilizza principalmente tecniche di computer vision, per cercare di riconoscere e classificare delle azioni compiute dall'uomo a partire da immagini o sequenze di immagini (video). In questo contesto, la *violence detection* si inserisce come un particolare caso di *action recognition*, infatti si tratta di un problema binario in cui si deve rilevare la presenza o meno di un comportamento violento all'interno di una sequenza di frame. Per identificare un atto violento abbiamo bisogno di tre informazioni fondamentali che prendono il nome di *informazioni spazio-temporali*:

- L'informazione spaziale: l'aggressione ha una particolare rappresentazione.
- L'informazione temporale: l'aggressione si sviluppa durante un periodo temporale ben determinato.
- L'informazione del movimento come l'accelerazione: l'aggressione è caratterizzata da brusche variazioni di velocità (i colpi).

Le tecniche utilizzate per la *violence detection* sono diverse, ma in questo lavoro di tesi ci limiteremo a considerare quelle basate sul *deep learning*. Diversi studi [4] [5] [6] [7] mostrano, infatti, come queste tecniche hanno raggiunto alte performance anche nel riconoscimento comportamentale, motivo per cui il lavoro di Accattoli [1] che è stato ripreso in questa tesi si era concentrato maggiormente su tali tecnologie.

2.2 - Deep learning applicato alla Violence Detection

Negli ultimi anni, con il grande successo del *deep learning* nell'ambito del riconoscimento comportamentale, molti studi hanno proposto l'utilizzo delle reti neurali per la *violence detection* [8] [9] [10] [11] [12] [13] [14].

Come sottolineato anche da Accattoli [1] nel suo lavoro, le strutture convoluzionali standard sono capaci di apprendere solo le informazioni spaziali, essendo state pensate per lavorare senza l'informazione temporale. Questo significa che l'intera informazione spazio-temporale non può essere appresa con un'architettura standard. Per ovviare a questo problema sono state definite nello stato dell'arte alcune soluzioni che potessero estrarre anche l'informazione temporale.

Gli approcci su cui si era focalizzata la tesi di Accattoli [1] erano divisi in due categorie:

- *Spatio-temporal Networks*. Come viene mostrato in [15] nelle reti spazio-temporali si cerca di riprodurre la struttura convoluzionale introducendo però anche l'informazione temporale. Tra gli approcci studiati troviamo due categorie facenti parte delle spatio-temporal network: 3D Convolutional Neural Networks e le Recurrent Neural Network [16]. Se le reti convoluzionali operano con delle immagini, attraverso operazioni di convoluzione e pooling bidimensionali, le 3D ConvNet prendono in ingresso una sequenza di frame processandoli attraverso filtri tridimensionali. Come mostrato in [17], esistono tre modelli per l'implementazione delle 3D ConvNet che definiscono come l'informazione temporale venga introdotta nella rete: early fusion, late fusion e slow fusion. Le RNN, invece, sono delle reti neurali in cui vengono permesse delle connessioni a retroazione, introducendo così una memoria.

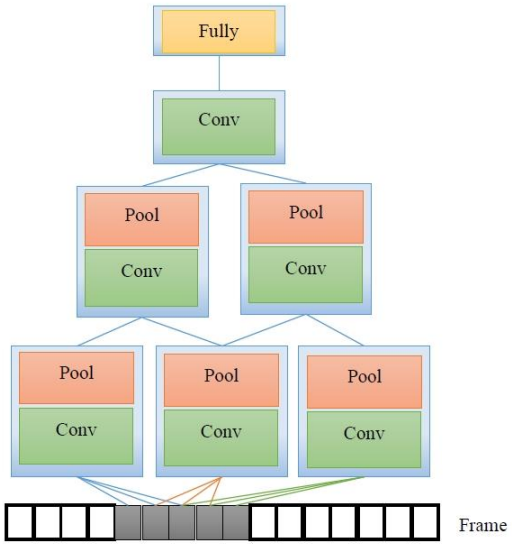

Figura 1 - Esempio di Spatio-temporal Network in modalità late-fusion tratto da [1]

- *Multiple Stream Networks*. Come illustrato nei lavori [11] [13], le reti neurali a stream multiplo sono rappresentate da più reti parallele (stream) che processano informazioni differenti, al fine di estrarre feature diverse (feature spaziali, temporali, ecc…). Ogni stream rappresenta un flusso di informazioni che, al termine dell'elaborazione, verrà combinato per produrre una rappresentazione univoca del video iniziale.

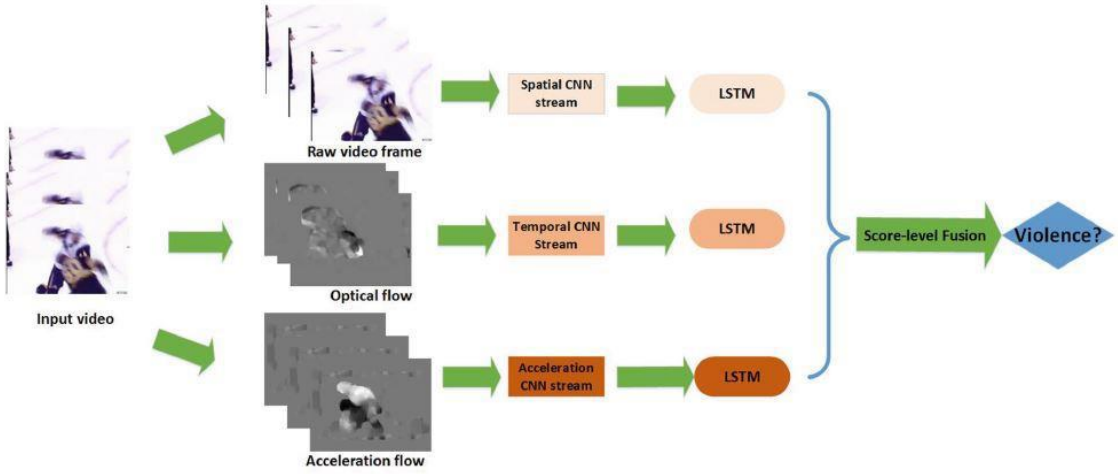

Figura 2 – Esempio di multi-stream neural networks tratto da [11]

In questo elaborato, a differenza di quanto fatto da Accattoli [1], tratteremo nel dettaglio solo la prima delle due categorie e di essa ci focalizzeremo soltanto sul primo approccio che è quello simile alla tecnica proposta in questo lavoro di tesi.

2.2.1 – Tecnica Spatio-temporal: 3D ConvNet

Ding et alii [14] propongono, come sottolineato da Accattoli [1] nel suo elaborato, l'utilizzo di una rete convoluzionale tridimensionale per il problema della *violence detection*. Come già accennato e vedremo poi, più nel dettaglio, nel prossimo capitolo, una 3D ConvNet non è altro che una rete convoluzionale standard in cui viene aggiunta la dimensione temporale. La rete illustrata in [14] lavora con un input di dimensione 60x90x90 ed è caratterizzata da 9 layer (compresi i layer di input e output). I tre layer finali sono rappresentati rispettivamente da due layer fully connected e un softmax layer, che si preoccupano di combinare le feature estratte dalla rete ed eseguire la classificazione del video in violento o non violento. I restanti layer, invece, sono ottenuti alternando uno strato Convoluzionale 3D con uno di Pooling 3D. Nella figura seguente viene rappresentata nel dettaglio l'architettura della rete, specificando la dimensione delle feature map ad ogni strato.

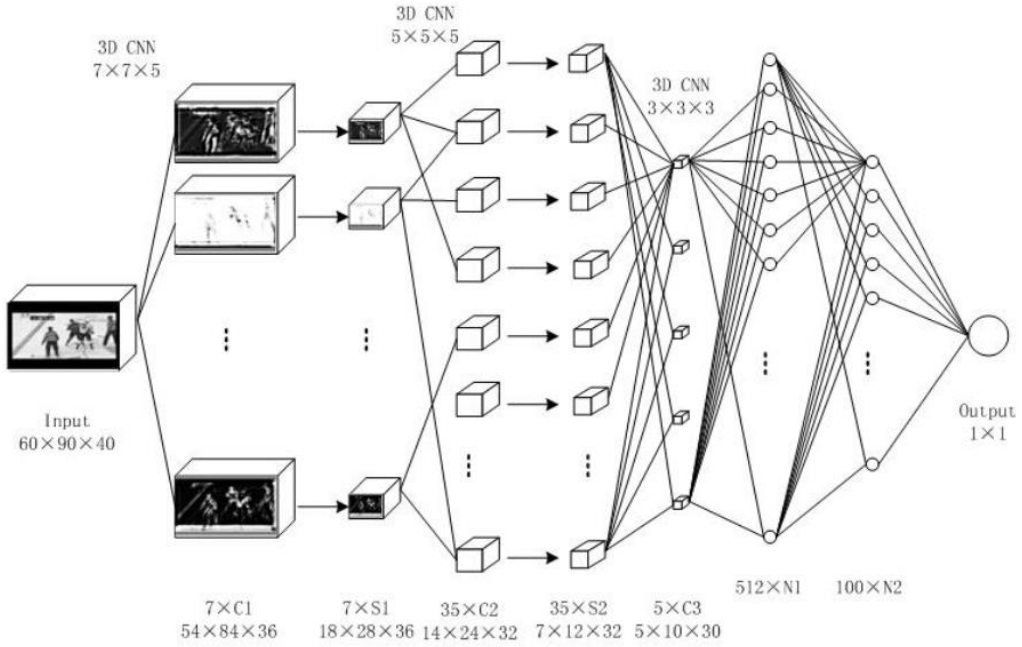

Figura 3 - Architettura rete proposta in [14]

Per addestrare il modello gli autori hanno utilizzato la *stochastic gradient descent*, che minimizza l'errore che intercorre tra l'output reale e quello prodotto dalla rete. L'addestramento viene eseguito direttamente sui dataset di benchmark.

Come già accennato precedentemente, questo approccio è simile alla tecnica proposta nel lavoro di tesi di Accattoli [1] che qui viene ripreso. Le differenze sostanziali del nostro modello rispetto a quello di Ding et alii sono:

- L'architettura della rete. Infatti, nonostante entrambe le soluzioni siano delle 3D ConvNet le architetture risultano differenti.
- Il modello proposto in questo elaborato e in quello di Accattoli [1] è pre-addestrato su un dataset differente.
- La nostra rete viene utilizzata come un estrattore di feature, perciò la classificazione non viene eseguita tramite una MLP (Multi Layer Perceptron), ma da un classificatore opportuno (SVM).

2.3 – Support Vector Machine (SVM)

Le Support Vector Machine (SVM) sono dei classificatori molto noti in letteratura per le buone performance che riescono ad ottenere [18]. Esse non sono altro che dei modelli supervisionati di Machine Learning usati per la classificazione o per la regressione. L'idea alla base delle SVM consiste nella ricerca di una determinata funzione di decisione, a partire da un insieme di classi e di

esempi etichettati. Questa funzione ha, poi, lo scopo di prevedere quale sia la classe di appartenenza di un ulteriore input non etichettato. Per spiegare il funzionamento di un classificatore basato sulle SVM riportiamo l'esempio illustrato da Accattoli nel suo elaborato [1] che espone alla perfezione il concetto. Consideriamo innanzitutto un dataset di esempio con due classi e due feature. Ogni campione del dataset sarà caratterizzato da dei valori in corrispondenza di ogni feature e un'etichetta della classe di appartenenza. Graficamente:

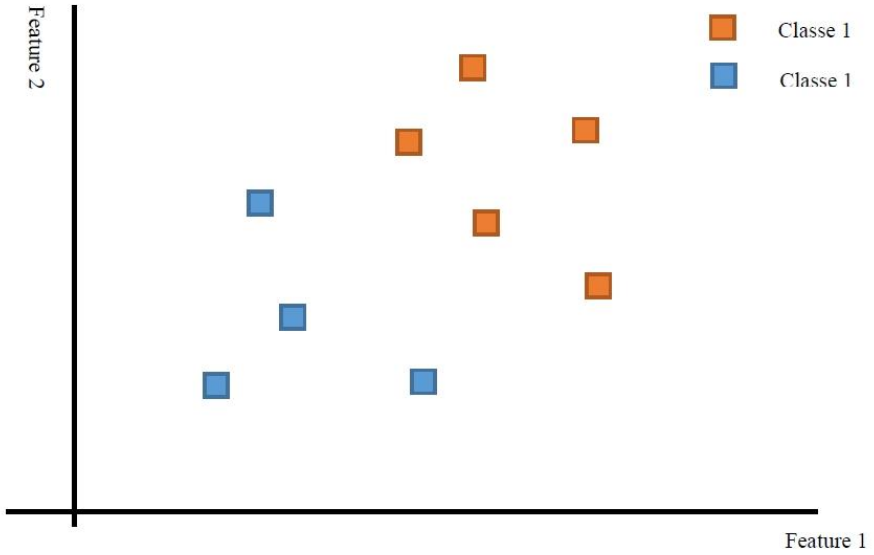

Figura 4 - Esempio di dataset di training tratto da [1]

Le Support Vector Machine a partire dai dati forniti, i cosiddetti dati di training, ricercano una retta che separi le due classi massimizzando il margine di distanza tra la retta e gli elementi delle classi. Tale retta viene definita *bordo di decisione.* Graficamente possiamo vederlo come segue:

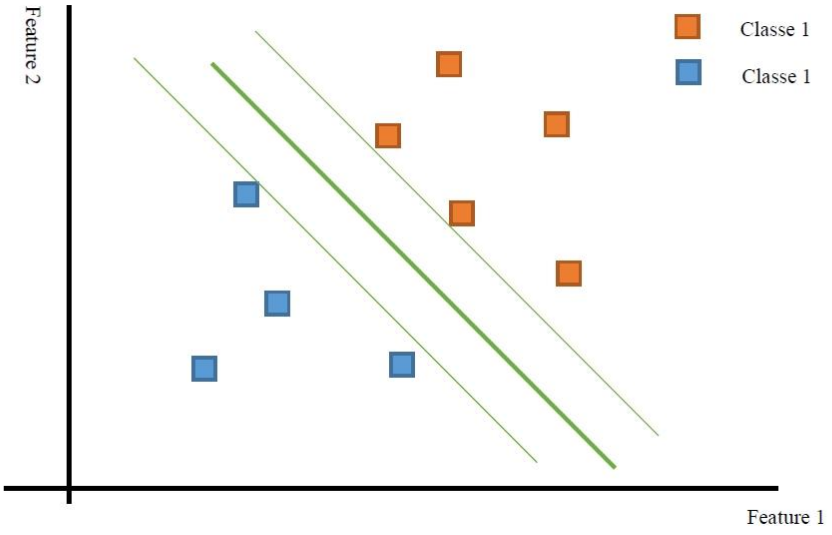

Figura 5 - Esempio di bordo di decisione tratto da [1].

In questa rappresentazione grafica, la retta verde più spessa rappresenta la funzione di decisione da trovare, mentre le due rette sottili rappresentano il margine che deve essere massimizzato. Come si può vedere attraverso il bordo di decisione si delineano due regioni distinte, una per ogni classe. La classificazione avviene poi associando una determinata classe ad un input non etichettato, in funzione della regione in cui ricade. Le Support Vector Machine che come in questo esempio utilizzano come funzione di decisione una retta, prendono il nome di SVM lineari o con kernel lineare.

2.4 – Motivazioni per la scelta del metodo utilizzato

Le motivazioni che avevano portato Accattoli nel suo lavoro [1], che è stato ripreso in questa tesi, ad optare per le tecniche basate sul *deep learning* sono diverse. Innanzitutto, nonostante questi approcci necessitino di dataset di esempi molto ampi per eseguire un training soddisfacente, hanno una migliore capacità di generalizzazione rispetto ad altri approcci utilizzati nello stato dell'arte e quindi sono capaci di ottenere buone performance sia nelle situazioni di aggressione in contesti affollati sia in quelle di violenza tra due individui. Oltre al fatto che possono essere applicate su ogni campo e riescono ad estrarre feature di alto livello raggiungendo così alte performance [19]. Inoltre, a differenza di altre tecniche riportate nello stato dell'arte come, ad esempio, quelle basate su rappresentazioni locali, le tecniche basate sul deep learning, una volta fissato l'output layer, producono delle feature di dimensione fissa. Infine, le reti neurali non hanno limitazioni tipiche di altri approcci come il problema dell'apertura e non hanno eventuali discontinuità o difficoltà di rappresentazione quando la telecamera è in movimento. Al contrario esistono studi che dimostrano come l'aggiunta di rumore all'interno dei dati di input limitano il fenomeno dell'overfitting e migliorano le performance [20].

In particolare, nel suo progetto Accattoli [1] tra le varie tecniche basate sul *deep learning* aveva deciso di usare una rete convoluzionale 3D per il problema della *violence detection*. Infatti, tra i vari vantaggi di questo approccio si ha che le reti convoluzionali 3D permettono di apprendere la rappresentazione spaziale e temporale direttamente utilizzando il video grezzo senza dover effettuare nessun pre-processing, oltre al fatto che non si ha bisogno di informazioni a priori sul problema.

Capitolo 3

Rete Neurale Convoluzionale a 3 dimensioni C3D

La rete neurale convoluzionale 3D che è stata usata in questo lavoro di tesi, come anche in quello di Accattoli [1], è un modello di rete pre-addestrato chiamato C3D progettato originariamente per l'*action recognition*, in particolare per il riconoscimento di quale sport sia rappresentato in un determinato video. Come vedremo poi dettagliatamente nel prossimo capitolo, quello dedicato ad illustrare il sistema sviluppato utilizzando nuove e più moderne tecnologie, abbiamo deciso di utilizzare C3D come un estrattore di feature, combinandolo poi con una SVM lineare per eseguire la classificazione dei video in violenti o meno.

Invece, in questo capitolo, illustreremo velocemente i concetti alla base del funzionamento delle reti convoluzionali riprendendo in maniera sintetica, come fatto anche in quello precedente, ciò che era stato messo in luce nel lavoro di tesi di Accattoli [1]. In particolare, verranno trattati la struttura di una rete convoluzionale standard e in cosa differisce da questa una rete convoluzionale 3D, per poi passare ad analizzare l'architettura con relativo funzionamento di C3D. La teoria introdotta in questo capitolo è ripresa dal libro "Deep Learning" [21] su cui l'elaborato di Accattoli [1] si basava.

3.1 – Architettura delle reti convoluzionali standard

L'idea alla base delle reti convoluzionali è quella di ridurre le connessioni rispetto alle normali reti neurali *Multi Layer Perceptron (MLP)*. Questo viene fatto introducendo nuovi layer che sostituiscono quelli fully-connected delle MLP, che vengono chiamati così perché tutti i neuroni di uno strato sono connessi con tutti i neuroni dello stato precedente generando però un enorme numero di connessioni che rendono in certe applicazioni impraticabile l'uso delle MLP. Come viene indicato in [21], il nome "*Convolutional*" delle *Convolutional Neural Network (CNN)* deriva dall'operazione matematica che viene ampliamente utilizzata: la convoluzione. Questa è un'operazione tra due funzioni di una variabile che consiste nell'integrare il prodotto tra la prima e la seconda traslata di un certo valore.

Come illustrato in [1], le caratteristiche delle *Convolutional Neural Network* che le differenziano dalle classiche *Multi Layer Perceptron* e che permettono di ridurre notevolmente il numero dei pesi sono:

- *Connettività Locale (Local Connectivity).* I neuroni di un determinato strato non sono connessi con tutti i neuroni dello strato precedente, ma solo ad una parte di essi. Questa regione di neuroni è un parametro specifico della rete e viene chiamato *receptive field.* In questo modo ogni neurone esegue un'elaborazione locale riducendo le connessioni che limiteranno notevolmente il numero di pesi da memorizzare.
- *Pesi condivisi (Weight Sharing).* Per ridurre ulteriormente il numero di pesi utilizzati essi vengono condivisi a gruppi. Questo implica che esistono delle connessioni con gli stessi pesi.

I pesi di una *CNN* sono i parametri allenabili della rete, che vengono utilizzati per il processamento dell'input, e sono rappresentati tramite l'uso di una matrice chiamata *kernel*.

I layer che compongono una rete convoluzionale sono i seguenti:

- Layer Convoluzionale
- Layer di Pooling
- Layer Fully-connected
- Layer Softmax

L'architettura di una rete convoluzionale non è altro che l'alternanza tra più di questi strati.

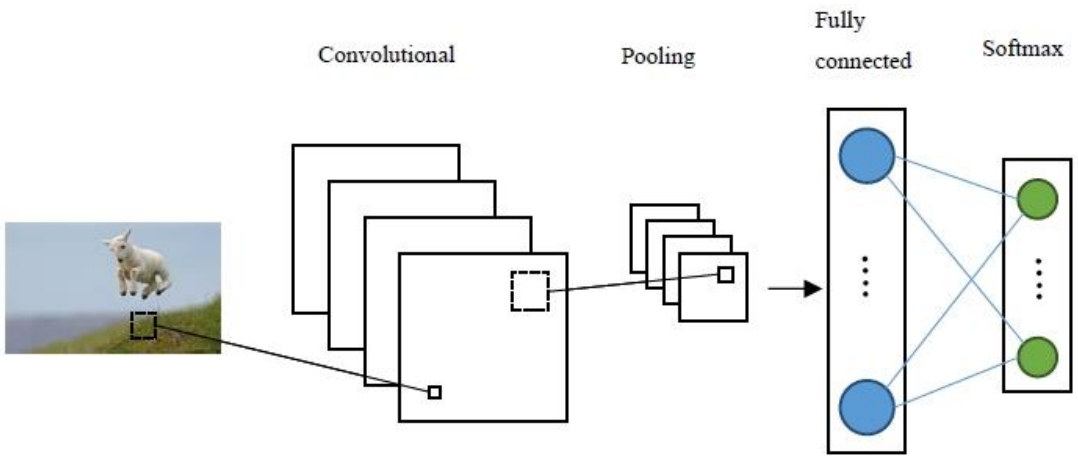

Figura 6 - Esempio architettura di una CNN tratto da [1]

3.1.1 – Layer Convoluzionale

Come si può facilmente intuire, questo è il layer fondamentale che caratterizza una rete convoluzionale. Il suo compito è quello di estrarre le feature a partire dall'informazione in input. L'estrazione viene eseguita applicando l'operazione di convoluzione ai dati in ingresso e le feature estratte dipendono dalla tipologia di kernel utilizzato. La convoluzione, inoltre, permette di preservare la relazione spaziale tra i pixel.

Il principio alla base del funzionamento del layer convoluzionale è quello di convolvere il kernel (definito sul layer) su tutto l'input fornito alla rete. Questa operazione produce come output una matrice che identifica la feature estratta tramite il kernel definito. Tale matrice prende il nome di *feature map* e può subire ulteriori elaborazioni dai layer successivi per apprendere ed estrarre feature sempre di più alto livello.

Ci sono, comunque, due cose importanti da sottolineare riguardanti il kernel utilizzato, perché mettono in luce il ruolo delle proprietà di connettività locale e dei pesi condivisi di cui parlavamo precedentemente. Innanzitutto, il kernel viene applicato non sull'intera immagine in input, ma solo in una sua regione (*local connectivity)*. Inoltre, lo stesso kernel viene applicato su tutti gli elementi in input (weight sharing).

Ci sono tre parametri fondamentali del layer convoluzionale che devono essere fissati e determinano la dimensione della *feature map*:

- *Depth.* Un filtro permette di produrre un'unica *feature map*, ma spesso si è interessati ad estrarre più feature dalla stessa immagine. Ciò viene fatto utilizzando più filtri applicati allo stesso input producendo ognuno una *feature map* differente. Questo parametro rappresenta proprio il numero di questi filtri e di conseguenza delle *feature map* estratte.
- *Stride.* Questo parametro rappresenta quant'è lo scorrimento applicato al kernel sull'elemento di input durante la convoluzione. Più lo stride è largo e più le *feature map* saranno di dimensioni ridotte.
- *Zero-padding*. Per poter catturare l'informazione ai lati dell'immagine è possibile applicare uno zero-padding ai bordi di essa.

In aggiunta a questi tre parametri un'altra caratteristica da fissare è la cosiddetta *filter size*, cioè la dimensione dei filtri.

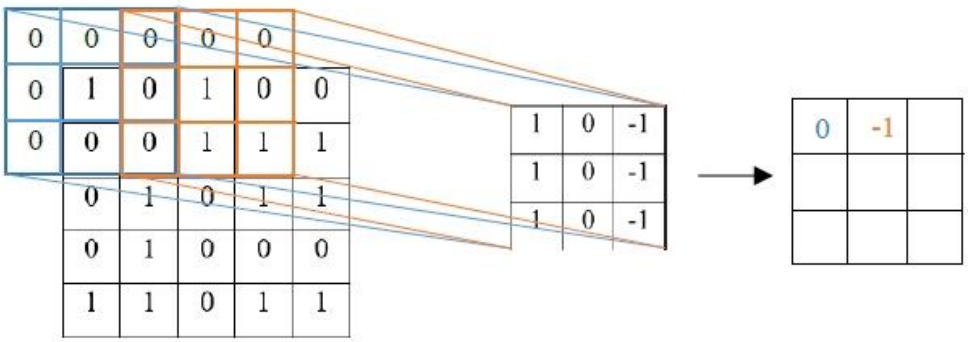

Figura 7 - Esempio di convoluzione con un singolo filtro, utilizzando lo zero-padding e con stride pari a 2 tratto da [1]

3.1.2 – Layer di Pooling

Lo scopo principale del layer di pooling è quello di ridurre la dimensione delle *feature map*, eliminando informazione irrilevante, ma mantenendo quella utile. In poche parole, il layer di pooling serve a ridurre progressivamente la dimensione della rappresentazione dell'input. Questo viene fatto applicando una funzione alla *feature map* chiamata appunto operazione di pooling. Non esiste una sola tipologia di operazione di pooling, ma ne esistono diverse in base a quale determinata funzione viene applicata. Tra le più note in letteratura troviamo il Max Pooling e l'Average Pooling.

Presa in input una *feature map*, il principio di funzionamento alla base di questo layer è quello di definire una finestra spaziale che scorre tutta la *feature map* e applicare l'operazione di pooling agli elementi dell'input che cadono all'interno di questa finestra. Ovviamente il risultato che otteniamo è ancora una volta una *feature map*, che rappresenta la stessa che avevamo in partenza, ma con dimensioni ridotte.

Come viene indicato in [21] e riassunto in [1], i vantaggi nell'utilizzo del layer di pooling sono i seguenti:

- Rendere le feature estratte invarianti alle trasformazioni affini come rotazioni, traslazioni, distorsioni, ecc… Ad esempio, un'eventuale piccola distorsione non modificherebbe l'output del Max Pooling, il cui principio di funzionamento è prendere il massimo tra i valori compresi all'interno della finestra spaziale, in quanto nello stesso intorno il valore più alto sarà sempre lo stesso.
- Diminuire il numero dei parametri e il tempo di computazione tramite la riduzione della dimensione delle *feature map*.

- Limitare il fenomeno dell'overfitting.
- Generare una rappresentazione ridotta e più semplice da utilizzare.

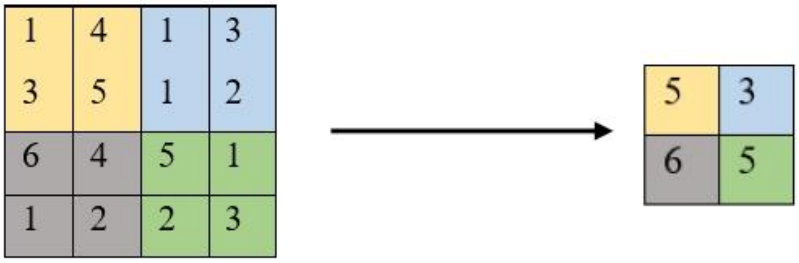

Figura 8 - Esempio di applicazione del Max Pooling con stride 2 e filtro di dimensione 2x2 tratto da [1]

3.1.3 – Layer Fully-connected e Softmax

Gli ultimi due layer vengono usati per eseguire la classificazione ed è interessante sottolineare come non siano stati introdotti dal modello convoluzionale. Infatti, il layer *fully-connected (FC)* non è rappresentato da altro che una *Multi Layer Perceptron* tradizionale. Questo layer viene combinato con una funzione di attivazione *softmax*, che permette di restituire un grado di affinità tra le classi utilizzate nell'addestramento della rete.

I due layer non fanno altro che prendere in ingresso le feature di alto livello estratte dagli strati precedenti ed eseguire la classificazione dell'input. Proprio per questa loro funzione, vengono posti in fondo alla rete. Le classi potrebbero essere calcolate direttamente dall'output convoluzionale, ma si è osservato che utilizzare i layer *fully-connected* permette di combinare le feature ottenendo risultati migliori.

In questi layer, a differenza dei precedenti, non dobbiamo specificare l'insieme dei parametri che abbiamo già analizzato prima, ma basta definire soltanto il numero di neuroni presenti nello strato. L'output prodotto, alla fine, sarà un vettore monodimensionale, proprio come avviene per le *Multi Layer Perceptron*. Questo, ovviamente, implica che dopo i layer *FC* non possono essere inseriti altri layer convoluzionali.

3.1.4 – Training

Infine, concludiamo questa trattazione sulle reti convoluzionali standard parlando dell'addestramento della rete. Questo risulta simile a quello che viene utilizzato per le reti neurali standard, tenendo presente, però, che viene modificato il normale algoritmo di apprendimento per adattarlo ai layer convoluzionali.

L'algoritmo utilizzato è lo *stochastic gradient descent*, già citato nel capitolo precedente, con la *backward propagation* [21]. I vari passi del processo di addestramento sono riassunti qui di seguito:

1. Si inizializzano i parametri dei kernel e dei pesi in maniera randomica.
2. Si esegue la *forward propagation*. La rete prende in ingresso un input etichettato con una classe che viene processato attraverso i pesi randomici e produce il grado di confidenza relativo alle classi da predire. Ovviamente con i pesi così assegnati è normale che l'output della rete sia totalmente errato.
3. Si calcola l'errore totale facendo riferimento all'output desiderato e quello ottenuto. Per fare questo è necessario definire la funzione che quantifica questo errore che viene denominata *loss function*. Tra le più note in letteratura si ha l'errore quadratico medio.
4. A questo punto si utilizza la *back propagation* per calcolare il gradiente dell'errore totale rispetto a tutti i pesi della rete propagandolo all'indietro tra i vari strati. Viene chiamata back propagation proprio perché il processo parte dallo strato di output e viene ripercorsa la rete all'indietro. Grazie al gradiente si aggiornano i pesi cercando di minimizzare la *loss function*.

I vari step da 2 a 4 vengono ripetuti per ogni elemento del training set.

3.2 – Reti convoluzionali 3D

Come era già stato anticipato nel capitolo precedente, per poter individuare un'aggressione dobbiamo considerare anche l'informazione temporale. Le reti convoluzionali standard che abbiamo però descritto finora non sono in grado di apprendere questa informazione, ma solo quella spaziale. Come viene indicato in [3], le *Convolutional Neural Network* perdono l'informazione temporale dell'input subito dopo ogni operazione di convoluzione e di pooling, anche nel caso in cui l'input fosse caratterizzato da frame consecutivi. Infatti, la sequenza di frame viene gestita in una rete convoluzionale tradizionale come un'immagine con differenti canali.

Come si può vedere nell'immagine sottostante tratta dall'elaborato di Accattoli [1], l'informazione temporale in input viene completamente collassata in una *feature map* bidimensionale, che per tale motivo ovviamente è capace di contenere solo l'informazione spaziale. Per impedire che l'informazione temporale vada persa dobbiamo fare in modo che l'output contenga anche la terza dimensione, il tempo, diventando così una *feature map* tridimensionale.

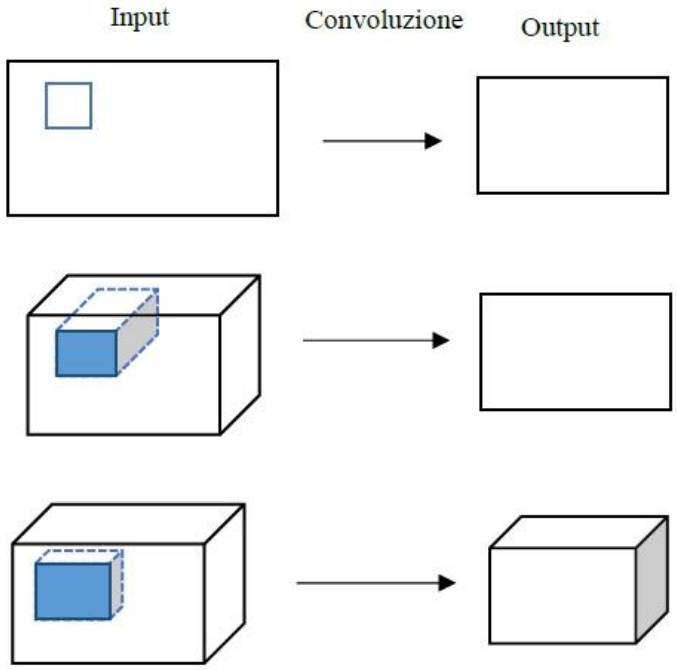

Figura 9 - Esempi di convoluzione 2D e 3D tratti da [1]
Partendo dall'alto troviamo una convoluzione 2D e un input rappresentato da una singola immagine. Al centro si hanno più frame in input per una convoluzione 2D. Infine, in basso viene rappresentata una convoluzione 3D.

Le reti convoluzionali 3D [3] (*3D ConvNet o CNN-3D*) non sono che l'estensione delle tradizionali 2D con l'introduzione della terza dimensione, quella temporale. Le *CNN-3D* hanno l'abilità di modellare il tempo attraverso le operazioni di convoluzione e di pooling 3D.

3.2.1 – Operazioni di Convoluzione e Pooling 3D

Le operazioni di convoluzione e di pooling 3D non sono altro che l'equivalente di quelle viste precedentemente, ma con l'aggiunta della dimensione temporale [3]. L'input di un layer 3D non è più rappresentato da una singola immagine, ma da una sequenza di immagini che contengono il movimento che vogliamo apprendere. Come abbiamo già anticipato, per fare in modo che l'output continui a mantenere l'informazione temporale questo deve essere tridimensionale. Ciò significa che

per poter estrarre delle feature in tre dimensioni bisogna modificare i filtri utilizzati nella convoluzione e nel pooling.

I filtri delle *CNN-3D* non sono più rappresentati da matrici ma da un cubo, anche in questo caso localmente connesso con una regione dell'input e con i pesi condivisi. L'operazione di convoluzione 3D viene eseguita su una sequenza temporale di frame, ovvero convolvendo un 3D kernel con il cubo formato impilando più frame contigui tra loro [6]. Come la sua controparte 2D, un singolo kernel 3D può estrarre un'unica feature, quindi un layer convoluzionale è formato dalla combinazione di più kernel 3D, estraendo così diverse *feature map*.

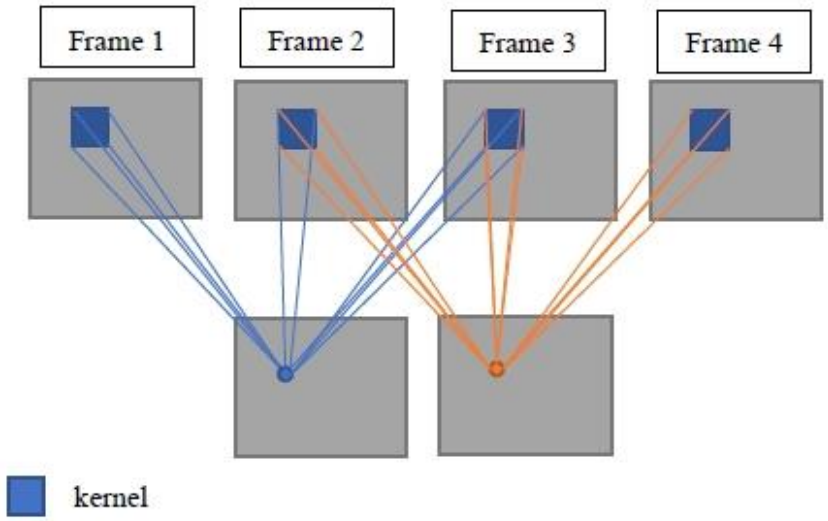

Figura 10 - Esempio di convoluzione 3D con 4 frame in input approssimato in 2D tratto da [1]

Per quanto riguarda, invece, l'operazione di pooling 3D, la somiglianza con la controparte 2D è ancora più marcata. Infatti, le funzioni applicabili sono le stesse ma con l'utilizzo di filtri tridimensionali.

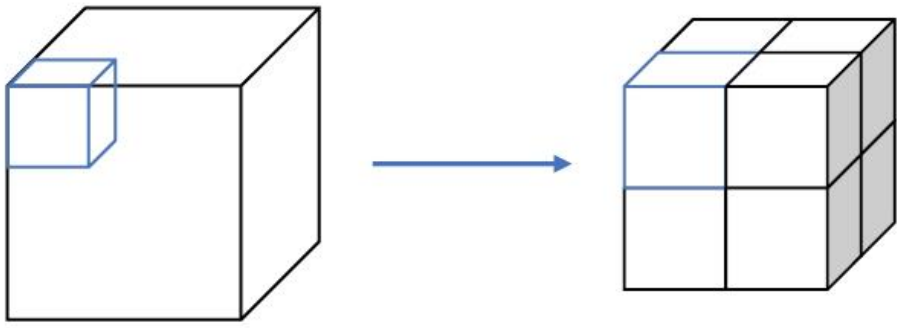

Figura 11 - Esempio di 3D Pooling tratto da [1]

3.3 – Modello di rete C3D

Come già ripetuto più volte, il modello di rete utilizzato in questo lavoro di tesi, come quello di Accattoli [1], si basa su una rete convoluzionale 3D. C3D [3] è un modello di rete pre-addestrato sviluppato dai ricercatori di Facebook, progettato per il problema dell'*action recognition*. Il modello è stato per prima cosa addestrato su un dataset di ridotte dimensioni molto conosciuto in letteratura, UCF101 [22], per poi essere riaddestrato su un dataset più ampio. Questo per un motivo ben preciso: l'addestramento è un'operazione molto pesante ed estremamente lenta, quindi per prima cosa si è cercata quale fosse l'architettura migliore su un dataset ridotto, per poi passare all'addestramento vero e proprio ottenendo così il modello finale della rete.

3.3.1 – Architettura di C3D

Alcuni studi [23] dimostrano come piccoli kernel di dimensione 3x3 producano risultati migliori per l'addestramento di reti neurali profonde. Questo ha portato i realizzatori di C3D ad usare kernel di questo tipo (*receptive field* 3x3), testando e modificando solo la profondità temporale dei kernel. L'architettura finale ottenuta dopo le prove su UCF101 è composta da 8 layer di convoluzione, alternati a 5 layer di pooling, seguiti da 2 layer fully-connected e infine un layer softmax finale. Come già accennato precedentemente in questo capitolo, gli strati finali (*FC e softmax*) sono utilizzati per eseguire la classificazione per il problema dell'*action recognition*. Proprio per questo motivo, nell'architettura implementata in questo lavoro di tesi si è deciso di non utilizzare questi layer, usando invece una *Support Vector Machine* per la classificazione.

La rete lavoro con input di dimensione 3x16xHxW, dove 3 sono i canali di colore, 16 la profondità temporale, mentre H e W rappresentano rispettivamente l'altezza e la larghezza del video. Si è scelto di lavorare con 16 frame consecutivi perché considerati un numero rappresentativo per identificare un movimento.

La dimensione dei kernel che ha riportato risultati migliori nei test è 3x3x3 con stride pari a 1x1x1. Nella figura sottostante possiamo vedere l'architettura di C3D in cui vengono riportate le dimensioni dei vari layer convoluzionali (il numero di filtri di ogni layer). Come si può facilmente notare, il numero dei filtri cresce progressivamente. Gli autori hanno dichiarato che questa tecnica permette di estrarre un numero di feature di alto livello maggiori a partire da limitate feature di basso livello.

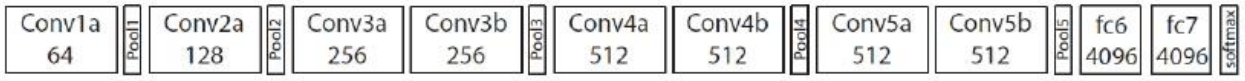

Figura 12 - Architettura C3D tratta da [3]

Gli strati di pooling, invece, utilizzano filtri di dimensione 2x2x2 con stride 2x2x2, ad esclusione del primo layer che una dimensione 1x2x2 e uno stride 1x2x2. Questa scelta è stata fatta per mantenere le informazioni temporali estratte negli strati iniziali. Per quanto riguarda l'operazione di pooling, gli autori hanno scelto di utilizzare il Max Pooling, visto che alcuni studi [24] hanno dimostrato come nel dominio del tempo il Max Pooling sia preferibile rispetto ad altre funzioni.

La funzione di attivazione dei neuroni, usata dalla rete, è la sigmoide, che limita i valori normalizzandoli tra 0 e 1. Infine, la dimensione dei layer fully-connected è di 4096. Ciò comporta che le feature estratte dai layer convoluzionali vengono combinate e rappresentate tramite un vettore di 4096 elementi.

3.3.2 – Addestramento della rete C3D

Per l'addestramento della rete è stato utilizzato uno dei più grandi dataset di benchmark per il riconoscimento su base video, Sport-1M, che è stato creato da una collaborazione tra Google e l'università di Stanford e contiene al suo interno più di un milione di video di carattere sportivo presi da Youtube.

Questo è stato uno dei motivi centrali nella scelta di questo modello per l'implementazione del sistema proposto nel lavoro di tesi di Accattoli [1] su cui si basa questo progetto. Infatti, come già accennato nel capitolo precedente, uno dei punti di debolezza delle reti neurali è la necessità di un addestramento su un dataset molto ampio per ottenere delle buone performance. Inoltre, grazie all'utilizzo di un dataset diverso per l'addestramento rispetto ai dati di lavoro si ottiene una migliore generalizzazione e si previene meglio l'overfitting [25] [26].

L'addestramento è stato eseguito inizialmente catturando due secondi in modo randomico da ogni video in quanto questi risultavano molto lunghi. Le clip estratte sono state in seguito ridimensionate a 128x171 ed infine ritagliate ad una dimensione di 2x16x112x112. Inoltre, il training è stato effettuato attraverso l'utilizzo dello *stochastic gradient descent*, già citato più volte in questo elaborato. Gli autori hanno scelto di utilizzare un *learning rate* variabile, inizialmente imposto a 0,003 e diviso per 2 ad ogni 150.000 iterazioni. Il processo di training è stato arrestato dopo 1.900.000 iterazioni complessive.

Capitolo 4

Il Sistema Realizzato

In questo capitolo, dopo esserci lasciati alle spalle i vari richiami teorici e aver introdotto la rete C3D, andremo ad illustrare dettagliatamente il sistema realizzato, che, come accennato nell'introduzione di questo elaborato, non è altro che un porting dell'architettura realizzata da Accattoli [1] utilizzando tecnologie più recenti. Inizieremo facendo una panoramica della piattaforma cloud utilizzata per implementare il nostro modello e delle varie librerie che sono state necessarie al nostro scopo. In seguito, proseguiremo mostrando nel dettaglio i quattro dataset di benchmark utilizzati per la *violence detection,* tre dei quali sono molto utilizzati in letteratura. Infine, vedremo come è stato realizzato il nuovo sistema di riconoscimento, sottolineando durante la trattazione quali sono le sostanziali differenze da quello passato.

4.1 – Tecnologie Utilizzate

Il sistema di riconoscimento è stato realizzato sulla piattaforma cloud di Google, *Google Colaboratory*, spesso abbreviata in "*Colab*" [27]. *Google Colab* permette agli utenti di scrivere ed eseguire codice Python direttamente nel browser con una serie di vantaggi: nessuna configurazione necessaria, accesso completamente gratuito alle GPU e condivisione semplificata. La versione di *Colab* utilizzata in questo lavoro di tesi aveva a disposizione 12,72 GB di RAM e 68,40 GB di spazio su disco che sono delle ottime caratteristiche, soprattutto considerato che è un servizio gratuito. Per l'esecuzione del codice, questa piattaforma di cloud sfrutta i famosi notebook *Jupyter* [28]. Questi non sono altro che documenti interattivi nei quali possiamo appunto scrivere ed eseguire il nostro codice. Entrando un po' più nel dettaglio, questi documenti permettono di suddividere il codice in numerose celle, ognuna delle quali può contenere anche del testo informativo. L'uso dei notebook *Jupyter* è molto semplice e comodo ed è molto popolare per chi si occupa di Data Science, Machine Learning e Deep Learning. Infatti, tramite un unico documento è possibile sia eseguire tutti i vari step di un processo di analisi o processing, sia descriverne il comportamento in linguaggio naturale. Di fatto, i notebook rappresentano un ottimo modo per spiegare o anche semplicemente mostrare come agisce un algoritmo. Come detto precedentemente, *Google Colab* lavora con codice Python e infatti questo, nella sua versione 3.6.9, è il linguaggio di programmazione usato per la realizzazione e

il testing del sistema di riconoscimento. Infatti, il Python è un linguaggio multi-paradigma molto utilizzato in letteratura con una grande disponibilità di librerie per il Machine Learning e il Deep Learning.

Oltre la piattaforma di cloud utilizzata, ci sono altre tecnologie software che sono state usate per questo porting che devono essere assolutamente citate. Si tratta principalmente di librerie di supporto per il Python che hanno permesso di implementare con tecnologie attuali la rete C3D, così come di effettuare il testing dell'architettura realizzata e il preprocessing dei dati. Queste librerie sono le seguenti: *Keras* [29], *Sklearn* [30] e *OpenCV* [31].

Keras è una libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello. Creata per permettere una rapida prototipazione di reti neurali profonde, si concentra sulla facilità d'uso, la modularità e l'estensibilità. Infatti, non a caso il suo motto è "*Simple. Flexible. Powerful.".* In questo lavoro di tesi, *Keras* è stato utilizzato per fare il porting dell'architettura di C3D, che era stata originariamente implementata su un vecchio framework per il Deep Learning, *Caffe* [32].

Sklearn, abbreviazione di *Scikit-learn*, è anch'essa una libreria open source per l'apprendimento automatico sviluppata per il linguaggio di programmazione Python, proprio come *Keras*. Questa libreria contiene algoritmi principalmente di classificazione, regressione e clustering (raggruppamento), ma ha anche numerose altre funzioni. *Sklearn*, in questo progetto, è stata utilizzata per effettuare l'addestramento e il testing del classificatore SVM.

OpenCV, abbreviazione di *Open Source Computer Vision Library*, è una libreria multipiattaforma nell'ambito della visione artificiale in tempo reale. Il linguaggio di programmazione usato per sviluppare questa libreria è il C++, ma è possibile interfacciarsi anche attraverso il C, Python e Java. *OpenCV* è stata fondamentale in questo lavoro per il preprocessing dell'input.

4.2 – Dataset

I tre dataset di benchmark molto utilizzati per il problema della *violence detection* in letteratura sono: Hockey Fight Dataset [33], Crowd Violence Dataset [34] e Movie Violence Dataset [33]. Proprio per essere diventati ormai uno standard per questo problema, Accattoli [1] aveva deciso di testare il suo sistema su questi tre dataset, così da avere un confronto diretto con gli approcci definiti nello stato dell'arte. In questo lavoro di tesi, si è deciso per quanto riguarda l'addestramento del classificatore di

riutilizzare, come Accattoli, questi tre famosi dataset, mentre per quanto riguarda la fase di testing, che verrà trattata nel dettaglio nel prossimo capitolo, si utilizzerà in aggiunta ai precedenti anche un quarto dataset creato dall'AIRTLab [35].

Come era stato sottolineato nel capitolo precedente, l'architettura proposta lavora con degli input da 16 frame ciascuno da cui estrae le feature. Però ovviamente i filmati presenti nei vari dataset hanno una lunghezza maggiore. Per questo motivo abbiamo diviso ogni video in segmenti da 16 frame etichettati ognuno come violento o non violento. Ogni feature estratta dalla rete viene poi utilizzata dal classificatore SVM per il riconoscimento. Per addestrare e testare il classificatore è stata utilizzata la tecnica della 5-fold cross validation, che verrà poi approfondita più avanti. Riportiamo qui di seguito una panoramica dei quattro dataset, così da poterli analizzare un po' più nel dettaglio.

4.2.1 – Hockey Fight Dataset

L'Hockey Fight Dataset è la principale base di dati per il problema della *violence detection*, utilizzato da tutti gli approcci dello stato dell'arte. Questo è composto da una collezione di filmati raccolti da partite di hockey della National Hockey League (NHL). Come viene detto in [33], vi sono la presenza di ben 1000 clip in totale, ognuna composta da circa 50 frame con una dimensione di 720x576 pixel. Il dataset è perfettamente bilanciato, infatti abbiamo esattamente 500 video di aggressioni e 500 video di comportamenti normali. Inoltre, il dataset è suddiviso in due singole classi (aggressione e non aggressione) ed ogni clip è stata etichettata con la classe di appartenenza. Attualmente non esiste ancora una base di dati fortemente etichettata di dimensione maggiore che raffiguri aggressioni tra singole persone. L'etichetta, comunque, è caratterizzata da una stringa che rispetta la seguente espressione regolare:

(fi |no)+([0-9]{1,3})+_xvid.avi

Dove i primi due caratteri rappresentano la classe etichettata: "fi" in caso di aggressione e "no" in caso di nessuna aggressione. I numeri, invece, indicano l'indice del video rispetto ad una classe. Infine, è importante sottolineare che tutti i video del dataset presentano background e soggetti simili nella scena ripresa.

4.2.2 – Crowd Violence Dataset

Il Crowd Violence Dataset [34] è composto da 246 video presi da Youtube, contenenti riprese reali di gruppi di persone a degli eventi. Infatti, come suggerisce il nome, in questo dataset, a differenza

del precedente, sono rappresentate scene di violenza in situazioni di affollamento e non tra singole persone. Principalmente sono riprese di tifosi ad alcune partite, sia in caso di violenza che no. Il grande vantaggio dietro l'utilizzo di video di questo tipo è quello di poter testare il sistema direttamente in condizioni reali, senza introdurre approssimazioni o bias di alcun tipo.

Questo dataset, come il precedente, risulta etichettato fortemente e bilanciato, infatti abbiamo esattamente 123 video con scene di violenza e 123 video con comportamenti normali. Tutti i filmati sono memorizzati con codifica mpeg e la durata di ognuno varia tra 1-6 secondi, con i frame ridimensionati a 320x240 pixel. In questo caso, però, i video non hanno una codifica nel nome, ma sono memorizzati in directory diverse.

È fondamentale testare gli approcci proposti in situazione di affollamento, infatti la sovrapposizione di più persone potrebbe rendere più difficile l'estrazione di particolari feature, privilegiandone altre. A dimostrazione di questo, spesso gli algoritmi che raggiungono un'alta accuratezza nel caso di singole persone, risultano non molto performanti in situazioni affollate e viceversa. Questo dataset, però, a differenza dell'Hockey Fight Dataset, ha purtroppo due problemi: i pochi campioni a disposizione e la qualità dei filmati.

4.2.3 – Movie Violence Dataset

Il Movie Violence Dataset è stato realizzato dagli stessi autori dell'Hockey Fight Dataset [33] ed è composto da 200 clip fortemente etichettate, tratte per la maggior parte da film di azione. Anche questo dataset risulta essere completamente bilanciato, infatti abbiamo 100 video con scene di violenza e 100 video con comportamenti normali.

Questo dataset è stato presentato come un'approssimazione di scene di violenza in situazioni reali per testare le capacità di generalizzazione degli approcci proposti. A differenza dei dataset precedenti, però, questo contiene clip di video di differente durata e risoluzione.

4.2.4 – AIRTLab Violence Dataset

Il Dataset per la *violence detection* dell'AIRTLab [35] è composto da 350 video fortemente etichettati in due classi di video: violenti e non violenti. I video non violenti contengono però spesso comportamenti (ad esempio abbracci, pacche sulla spalla, ecc…) che possono causare dei falsi positivi, a causa dei movimenti veloci e delle somiglianze con alcuni comportamenti violenti.

A differenza dei dataset precedenti, questo dataset, però, non è bilanciato, infatti abbiamo 120 video raffiguranti comportamenti normali e 230 video raffiguranti scene di violenza. Inoltre, a loro volta le clip di ogni classe sono divise, questa volta in modo bilanciato, in due sottocartelle (cam1 e cam2) che contengono le stesse clip ma registrate da due differenti punti di vista e con due fotocamere diverse. Tutti i filmati sono memorizzati con codifica MP4 e la loro lunghezza media è di 5,63 secondi, con i filmati più corti che durano solo 2 secondi e i più lunghi che possono arrivare anche a 14 secondi. Inoltre, la risoluzione dei vari video è di 1920x1080 pixel e il frame rate è di 30 fps.

Tutte le clip sono state registrate nella stessa stanza con illuminazione naturale. Come accennato precedentemente, per ogni classe ognuna delle clip è registrata da due punti diversi della stanza e con diverse fotocamere. Queste sono rispettivamente la fotocamera del cellulare Asus Zenfone Selfie ZD551KL e la fotocamera TOPOP Action Cam OD009B. L'Asus Zenfone era posto nell'angolo in alto a sinistra della stanza di fronte alla porta, mentre la TOPOP Action Cam era posizionata nell'angolo in alto a destra della stanza sul lato della porta.

Come già accennato precedentemente, questo dataset verrà utilizzato solo nella fase di testing durante gli esperimenti insieme ai precedenti, mentre non comporrà il dataset intero definitivo usato per l'addestramento del classificatore SVM.

4.3 – Sistema di Riconoscimento

Per realizzare il sistema di riconoscimento sono stati tre gli elementi principali su cui ci siamo dovuti concentrare:

- *Preprocessing dell'Input*, illustrare cosa il sistema prende in input e come lo processa;
- *Feature extractor*, specificare come vengono estratte le feature da ogni input e soprattutto quali estrarre;
- *Classificatore*, definire come eseguire la classificazione data la rappresentazione estratta da ogni video.

Andiamo ora ad analizzare dettagliatamente queste tre componenti del sistema di riconoscimento.

4.3.1 – Preprocessing dell'Input

Come già affermato più volte, la rete C3D processa input di 16 frame e per questo si è deciso di suddividere ogni video in segmenti di tale dimensione. Però, come abbiamo visto, spesso i filmati dei vari dataset di benchmark sono di lunghezze differenti, quindi bisogna decidere come sfruttare

l'input. Inoltre, all'interno di uno stesso video la violenza non è per forza continua, ma potrebbero esserci delle pause tra una scena violenta e un'altra. Ciò implica che potrebbero esserci alcuni gruppi di frame all'interno dei filmati etichettati come violenti in cui, però, effettivamente la violenza non è presente.

Gran parte degli approcci usati in letteratura producono l'input attraverso un campionamento dei frame nel video, partendo da un istante casuale e campionando sequenzialmente. In questo lavoro di tesi, abbiamo ripreso la tecnica utilizzata da Accattoli [1] nel suo progetto. Infatti, abbiamo deciso anche qui di sfruttare tutto il filmato, così facendo si ottiene un input di dimensione maggiore e si evita il rischio di scartare informazioni rilevanti. A tale scopo ogni filmato viene diviso in modo sequenziale e non sovrapposto in segmenti da 16 frame, etichettati con la stessa label del video. Il dataset che si ottiene dall'unione di questi segmenti rappresenta una nuova base di dati utilizzata per l'addestramento e il testing del classificatore SVM. Ovviamente i video non saranno quasi mai un perfetto multiplo di 16 frame, quindi si è scelto di scartare i frame finali, nonostante questa decisione potrebbe sembrare andare contro alla volontà di non eliminare informazione. Però, in realtà, analizzando attentamente il dataset, si può osservare come gli ultimi frame spesso contengano informazioni inutili.

I frame raggruppati in segmenti verranno etichettati solo dopo aver estratto tutte le rappresentazioni dei tre dataset di benchmark famosi in letteratura. Ogni vettore di feature viene memorizzato opportunamente in un determinato file txt (aggressione o nessuna_aggressione) in funzione della classe di appartenenza e una volta completati i due file con le feature estratte da tutti i video dei tre dataset, l'etichetta viene associata alle varie feature andando ad inserirla come ultima colonna di ogni vettore in entrambi i file a seconda sempre della classe di appartenenza.

4.3.2 – Feature Extractor

L'elemento centrale del sistema di riconoscimento è l'estrattore di feature. Come ampiamente illustrato nel capitolo precedente, si è deciso di utilizzare a tale scopo un modello di rete neurale 3D pre-addestrato, chiamato C3D. Questo modello, però, è stato sviluppato per il problema dell'*action recognition*, quindi per utilizzare la rete come un estrattore di feature abbiamo dovuto apportare delle modifiche alla tradizionale architettura di C3D. In particolare, si è preso come output della rete quello del primo layer fully-connected (denominato fc6). Questo vettore monodimensionale rappresenta la combinazione delle feature estratte dalla rete ed ha una dimensione pari a 4096 elementi.

L'eventuale output prodotto da un layer FC successivo porterebbe sicuramente ad una rappresentazione peggiore, perché la rete è stata addestrata, come accennato già in precedenza, su video di carattere sportivo, che sono contraddistinti da comportamenti diversi da un atto violento. La violenza, infatti, spesso consiste in una collezione di calci e pugni senza un preciso schema, a differenza della maggior parte degli sport. Pertanto, qualsiasi elaborazione successiva tenderà ad assimilare l'input ad una delle classi sportive definite nel training set della rete neurale, portando così ad un peggioramento notevole delle performance.

4.3.3 – Classificatore

Le feature estratte da ogni blocco di 16 frame vengono poi passate al classificatore per eseguire il riconoscimento di una potenziale aggressione. Come già ampiamente discusso nei capitoli precedenti, il classificatore che è stato scelto per tale scopo è una SVM lineare. I principali vantaggi delle *Support Vector Machine* [18] possono essere riassunti come segue:

- Hanno delle ottime performance;
- Soffrono in modo limitato di overfitting;
- Una volta addestrate offrono alte velocità di classificazione;
- Non serve conoscere a priori la distribuzione dei dati.

Al contrario, le SVM non risultano così performanti quando ci troviamo ad affrontare problemi multi-classe e quando il training set non è linearmente separabile. Infatti, per quanto riguarda il primo di questi due svantaggi, per eseguire una classificazione multipla sono necessarie delle modifiche che rendono il classificatore più lento nel predire la classe di appartenenza.

Dopo aver analizzato vantaggi e svantaggi di questa tecnologia, la scelta di Accattoli [1] che qui è stata ripresa era ricaduta su una SVM lineare per due motivi fondamentali:

- Il problema della *violence detection* risulta essere binario e quindi non ci sono complicazioni introdotte da un sistema multi-classe;
- Il vettore di input ha una dimensione elevata (4096 elementi), quindi le feature risultano essere abbastanza discriminanti per ottenere buone performance anche con un kernel lineare.

Dopo aver costruito il dataset totale con le feature estratte da tutti i video dei tre dataset di benchmark famosi in letteratura, si passa all'addestramento del classificatore andando a divedere opportunamente il dataset intero in training set, che è composto dal 70% dei dati, e test set, che è

composto dal restante 30%. Una volta concluso l'addestramento, per ogni nuovo vettore di feature fornito in input al classificatore, questo produrrà in output la classe predetta tramite una label: 1 in caso di aggressione o 2 per l'assenza di violenza.

4.3.4 – Implementazione in Google Colab

L'architettura finale dell'intero sistema di riconoscimento può essere rappresentata come segue.

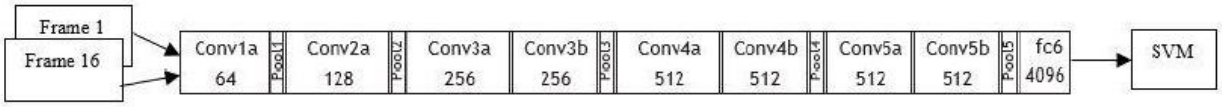

Figura 13 - Architettura del sistema di riconoscimento tratta da [1]

Di seguito, invece, andiamo ad analizzare nel dettaglio come è stato implementato l'intero sistema di riconoscimento su *Google Colab*.

Per quanto riguarda il modello di rete C3D, è stato effettuato un porting su *Keras* della sua vecchia versione rilasciata dai suoi autori su *Caffe*, che era stata usata da Accattoli [1] nel suo lavoro. Questo porting è basato su una repository Github [36]. In questa repository erano presenti l'implementazione dell'architettura di rete in due diverse varianti (in una vengono utilizzate le API funzionali di *Keras* mentre nell'altra quelle sequenziali) e i pesi ottenuti tramite l'addestramento sul dataset originale di Sport-1M, ovviamente aggiornati per essere compatibili con la versione 2.2.4 di *Keras* usata in questo progetto. Ovviamente questa architettura, come già affermato precedentemente, viene utilizzata solamente come estrattore di feature, quindi gli sono state apportate delle modifiche durante lo sviluppo del sistema di riconoscimento. In particolare, l'output deve essere rappresentato da un vettore di feature e non da un grado di confidenza; l'input è caratterizzato da un intero filmato diviso in segmenti e non solamente da 16 frame e la classificazione deve essere eseguita da un classificatore lineare SVM e non da una MLP. Riportiamo nella pagina seguente il codice relativo al modello di rete C3D implementato in *Keras,* nella sua variante che utilizza le API sequenziali e la funzione che genera l'estrattore di feature a partire dal modello C3D.

```
model = Sequential()
input_shape = (16, 112, 112, 3)

model.add(Conv3D(64, (3, 3, 3), activation='relu',
                 padding='same', name='conv1',
                 input_shape=input_shape))
model.add(MaxPooling3D(pool_size=(1, 2, 2), strides=(1, 2, 2),
                       padding='valid', name='pool1'))
# 2nd layer group
model.add(Conv3D(128, (3, 3, 3), activation='relu',
                 padding='same', name='conv2'))
model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2, 2, 2),
                       padding='valid', name='pool2'))
# 3rd layer group
model.add(Conv3D(256, (3, 3, 3), activation='relu',
                 padding='same', name='conv3a'))
model.add(Conv3D(256, (3, 3, 3), activation='relu',
                 padding='same', name='conv3b'))
model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2, 2, 2),
                       padding='valid', name='pool3'))
# 4th layer group
model.add(Conv3D(512, (3, 3, 3), activation='relu',
                 padding='same', name='conv4a'))
model.add(Conv3D(512, (3, 3, 3), activation='relu',
                 padding='same', name='conv4b'))
model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2, 2, 2),
                       padding='valid', name='pool4'))
# 5th layer group
model.add(Conv3D(512, (3, 3, 3), activation='relu',
                 padding='same', name='conv5a'))
model.add(Conv3D(512, (3, 3, 3), activation='relu',
                 padding='same', name='conv5b'))
model.add(ZeroPadding3D(padding=((0, 0), (0, 1), (0, 1)), name='zeropad5'))
model.add(MaxPooling3D(pool_size=(2, 2, 2), strides=(2, 2, 2),
                       padding='valid', name='pool5'))
model.add(Flatten())
# FC layers group
model.add(Dense(4096, activation='relu', name='fc6'))
model.add(Dropout(.5))
model.add(Dense(4096, activation='relu', name='fc7'))
model.add(Dropout(.5))
model.add(Dense(487, activation='softmax', name='fc8'))
```

```
def create_feature_descriptor(C3D_model, layer_name='fc6'):
    extractor = Model(inputs=C3D_model.input,
                      outputs=C3D_model.get_layer(layer_name).output)
    return extractor
```

Figura 14 - Codice relativo all'implementazione della rete C3D in Keras e dell'estrattore di feature

In questo progetto, le fasi di preprocessing dell'input e di estrazione delle feature vengono realizzate in un'unica funzione chiamata *preprocess_input_and_feature_extraction*, che prende come parametri l'estrattore di feature creato precedentemente, il *path* del video di cui si vogliono estrarre le feature e il nome del file txt in cui si desidera che i vettori di feature siano salvati. Con l'ausilio delle funzioni della libreria *OpenCV* si calcolano i frame totali del video da cui, poi, tramite una divisione intera per 16, si individuano il numero di segmenti in cui verrà diviso il filmato. A questo punto si procede ad estrarre e memorizzare in una lista chiamata *start_frames* i frame iniziali di ogni segmento, per poi creare effettivamente a partire da questa lista i segmenti di frame, come si può vedere in questa porzione di codice:

```
for start_frame in start_frames:
  # move to start_frame
  if int(major_ver)  < 3 :
    video.set(cv2.cv.CV_CAP_PROP_POS_FRAMES, start_frame)
  else:
    video.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

  # grab each frame and save in a numpy array
  for frame_count in range(num_frames_per_clip):
    frame_num = frame_count + start_frame
    print("[Info] Extracting frame num={}".format(frame_num))
    ret, frame = video.read()
    if not ret:
        print("[Error] Frame extraction was not successful")
        sys.exit(-7)
    videoFrames.append(cv2.resize(frame, (112, 112)))

v = np.array(videoFrames, dtype=np.float32)
```

Figura 15 - Codice relativo all'estrazione dei frame per creare i vari segmenti

Con questo si conclude la parte di preprocessing dell'input e quindi si passa ad estrarre le feature e ad inserirle in un file txt con il nome che abbiamo passato come parametro alla funzione. Quest'ultimo passaggio viene eseguito tramite la seguente porzione di codice:

```
with open(file_txt, 'ab') as f:
  for i in range(iteration):
      X = v[i*16:i*16+16]
      out = featureExtractor.predict(np.array([X]))
      print("Frames {} {}".format(i*16, i*16 + 15))
      print("Lenght {}".format(out.size))
      np.savetxt(f, out)
```

Figura 16 - Codice relativo all'estrazione e salvataggio su file delle feature

Infine, andiamo a vedere come viene eseguita la classificazione. Il classificatore addestrato viene caricato e gli vengono passate in input le feature estratte dal video che vogliamo classificare come violento o meno, salvate dentro il file txt che avevamo generato nel passo precedente. Il classificatore, a questo punto, restituisce tramite l'utilizzo del metodo *predict* un vettore con tante label quanti erano i segmenti in cui era stato diviso il filmato. Come già accennato precedentemente, queste label, che rappresentano la classe predetta, possono assumere i valori 1 o 2, che corrispondono rispettivamente alla presenza di aggressione o meno. Tutto questo viene implementato tramite la seguente porzione di codice:

```
clf = joblib.load('model.pkl')

feat = np.loadtxt(output_file, delimiter=" ")

pred_array = clf.predict(feat)
print(pred_array)
```

Figura 17 - Implementazione del processo di classificazione di un video

Come aveva fatto già Accattoli [1], anche in questo lavoro di tesi il video viene considerato come violento se almeno un segmento di 16 frame viene etichettato come tale.

Quello appena illustrato è il sistema di riconoscimento di scene di violenza all'interno di un singolo video e quindi, in quanto tale, lavora solo con un unico filmato. Per addestrare il nostro classificatore ed eseguire gli esperimenti che analizzeremo nel prossimo capitolo, si è dovuto ricorrere ad un'implementazione alternativa, che si diversifica da ciò che abbiamo visto finora per i seguenti punti che sono quasi gli stessi della tesi di Accattoli [1] per cercare di rimanere il più possibile fedeli al suo metodo di addestramento nonostante i diversi strumenti utilizzati:

- Innanzitutto, l'input è rappresentato dal dataset intero, che ricordiamo essere composto dai tre dataset di benchmark celebri in letteratura per la *violence detection*, e non da un solo video. Esso è stato diviso in due directory, una contenente tutti i video con scene di violenza e l'altra tutti quelli con comportamenti normali.
- L'estrazione delle feature viene eseguita per ogni filmato, salvando i vettori di feature di tutti i numerosi video in un unico file txt. In questo modo, abbiamo prodotto il dataset da utilizzare per il training e il testing del classificatore SVM.

- Ogni vettore di feature deve essere etichettato. Infatti, a differenza del caso precedente in cui il sistema veniva utilizzato per predire, durante l'addestramento è necessario associare ad ogni input la classe di appartenenza secondo la tecnica dell'apprendimento supervisionato. Si è deciso di implementare questa tecnica separando inizialmente in due file txt differenti i vari vettori di feature a seconda della classe di appartenenza, per poi effettuare il labeling durante la creazione del file txt definitivo di cui si è parlato al punto precedente. Ricordiamo che l'aggressione viene identificata con la label 1, mentre l'assenza di aggressione con la label 2. Questo passaggio è svolto dalla funzione chiamata *write_dataset*, il cui codice viene mostrato di seguito:

```
def write_dataset(source, destination, label):
  try:
    # apro il file delle features e ne leggo il contenuto
    fp = open(source)
    text = fp.read()

    # divido il contenuto per riga ottenendo una lista di features
    features = text.split('\n')

    for feature in features[:-1]:
      with open(destination, 'a') as f:
        f.write(feature + " " + str(label) + '\n')

  finally:
    fp.close()
```

Figura 18 - Implementazione della funzione write_dataset

- Il classificatore deve essere addestrato e testato. A partire dal dataset creato si costruiscono training e test set tramite uno splitting casuale. La dimensione del training set è del 70% dei dati, mentre quella del test set copre il restante 30%. L'addestramento del classificatore, a questo punto, viene eseguito attraverso la tecnica della 5-fold cross validation sfruttando i metodi messi a disposizione dalla libreria *Sklearn*. In particolare, come è possibile vedere nella figura sottostante, il classificatore di cui salviamo il modello è quello che tra i cinque restituiti in output dal metodo *cross_validate* di *Sklearn* ottiene l'accuratezza maggiore.

```
# Addestramento classificatore SVM

import pandas as pd
import numpy as np
import sklearn
from sklearn import svm
from sklearn.model_selection import ShuffleSplit, cross_validate
from sklearn.metrics import check_scoring
from sklearn.externals import joblib


X, y = load_dataset("full_dataset.txt")

# Cross Validation
clf = svm.SVC(kernel='linear', C = 1, probability=True)

cv = ShuffleSplit(n_splits=5, test_size=0.3)

scorer = check_scoring(clf, scoring=None)

cv_results = cross_validate(estimator=clf, X=X, y=y, groups=None,
                            scoring={'score': scorer}, cv=cv,
                            n_jobs=None, verbose=0, fit_params=None,
                            pre_dispatch='2*n_jobs', error_score=np.nan,
                            return_estimator=True)

scores = cv_results['test_score']
clfs = cv_results['estimator']

print(scores)
print("Accuracy: {0} +/- {1}".format(scores.mean(), scores.std() * 2))

i_max = np.argmax(scores)
print("Max index of scores array: {}".format(i_max))
final_clf = clfs[i_max]
joblib.dump(final_clf, 'model.pkl')
```

Figura 19 - Addestramento del classificatore SVM tramite l'utilizzo della 5-fold cross validation

Capitolo 5

Esperimenti e Risultati

Il seguente capitolo illustra i vari test effettuati sui quattro dataset di benchmark. In particolare, andremo a confrontare il sistema realizzato con alcuni approcci promettenti dello stato dell'arte e ovviamente con i risultati che aveva ottenuto Accattoli [1] nel suo lavoro di tesi. Il capitolo si concluderà con un'attenta analisi riguardante i risultati ottenuti e gli errori commessi dalla rete.

5.1 – Setting Sperimentale

Per valutare le performance del sistema realizzato, si è deciso di riutilizzare come metriche di valutazione le stesse che aveva utilizzato anche Accattoli [1] nel suo progetto per poter effettuare un confronto equilibrato, ovvero l'accuratezza e l'area under the curve (AUC), sfruttando a tal proposito, come già abbiamo avuto modo di vedere precedentemente, lo schema della 5-fold cross validation, così da rimanere conforme agli approcci appartenenti allo stato dell'arte e alla tesi più volte citata in questo elaborato. Il principio alla base della 5-fold cross validation è molto semplice: ogni dataset viene diviso in cinque differenti split, dove quattro dei quali sono utilizzati per il training e il restante per il testing. Questo viene poi ripetuto cambiando il test set per ognuno dei cinque split.

Anche nel nostro caso, il caricamento in memoria del dataset risultava essere ordinato per classe e quindi è stato necessario utilizzare lo shuffling, tramite la classe *ShuffleSplit* definita nella libreria *Sklearn*. Senza questa tecnica, infatti, gli split prodotti sarebbero risultati molto sbilanciati, impedendo di eseguire un addestramento corretto e riducendo, così, notevolmente l'accuratezza.

Attraverso la K-fold, l'accuratezza viene calcolata ad ogni iterazione e il risultato finale non è altro che la media dei vari risultati con relativa deviazione standard. Oltre a questo parametro, si è utilizzata come ulteriore metrica di valutazione, l'area sotto la curva ROC (AUCROC). L'AUC permette di osservare un confronto più diretto tra il rate dei casi classificati come veri positivi e quelli classificati come falsi positivi, che rappresentano i cosiddetti errori di tipo 1.

Il confronto nel Movie Violence Dataset, come anche nell'elaborato di Accattoli [1], non viene riportato in quanto è caratterizzato da un numero estremamente limitato di campioni, ma soprattutto

questi risultano molto facili da discriminare. Inoltre, visto che, come andremo ad analizzare tra poco, i nostri esperimenti riportano risultati che possono essere considerati praticamente uguali a quelli ottenuti da Accattoli sull'Hockey Fight Dataset e sul Crowd Violence Dataset, abbiamo preso per assodato che anche in questo caso il metodo proposto avrebbe ottenuto come il suo circa il 100% di accuratezza con una deviazione standard praticamente nulla. Oltretutto questo dataset è anche poco utilizzato.

5.2 – Test Hockey Fight Dataset

Nella tabella sottostante sono riportati il confronto tra il nostro metodo rispetto ad alcuni approcci promettenti dello stato dell'arte e all'architettura di Accattoli.

Existing	Algorithm	ACC ± SD	AUC
	LHOG + LHOF + BoW	95,1 ± 1,15 %	0,9798
	Three streams + LSTM	93,9 %	-
	CNN + LSTM	97,1 ± 0.55 %	-
	Two-stream + IDT	98,6 %	-
	MoSIFT + KDE + SC	94,3 ± 1,68 %	0,9708
	DIMOLIF	88,6 ± 1,2 %	0,9323
	3D Conv Net	91 %	-
	Accattoli's C3D	**98,51 ± 1,05%**	**0,9832**
Proposed	**C3D**	**97,61 ± 0,95%**	**0.9958**

Tabella 1 - Confronto di classificazione nell'Hockey Fight Dataset

Come si può osservare dalla tabella, la soluzione proposta raggiunge un'accuratezza elevata, ma comunque leggermente più bassa di quella di Accattoli e dell'approccio presentato in [13]. Per quanto riguarda quest'ultimo, gli autori hanno specificato però, che tale valore non è ottenuto tramite cross validation, ma è il loro risultato migliore raggiunto. Per quanto riguarda, invece, il confronto con i risultati ottenuti da Accattoli, questi si possono considerare praticamente dello stesso livello di

quelli che abbiamo ricavato per diversi fattori. Innanzitutto, la differenza nella media dell'accuratezza è meno di un punto percentuale, per di più bisogna tenere in considerazione che non possiamo sappere se i pesi di C3D siano esattamente gli stessi dell'architettura utilizzata da Accattoli. Infine, un altro fattore ancora più importante consiste nel fatto che gli split casuali non potranno mai essere identici a quelli usciti ad Accattoli durante i suoi esperimenti e quindi questo potrebbe influenzare molto i risultati.

Di seguito riportiamo anche la matrice di confusione relativa ad un singolo test con split casuali. Come nel caso di Accattoli, anche qui lo split prevedeva, come già illustrato ad inizio capitolo, l'utilizzo del 70% dei campioni per il training e del restante 30% per il testing.

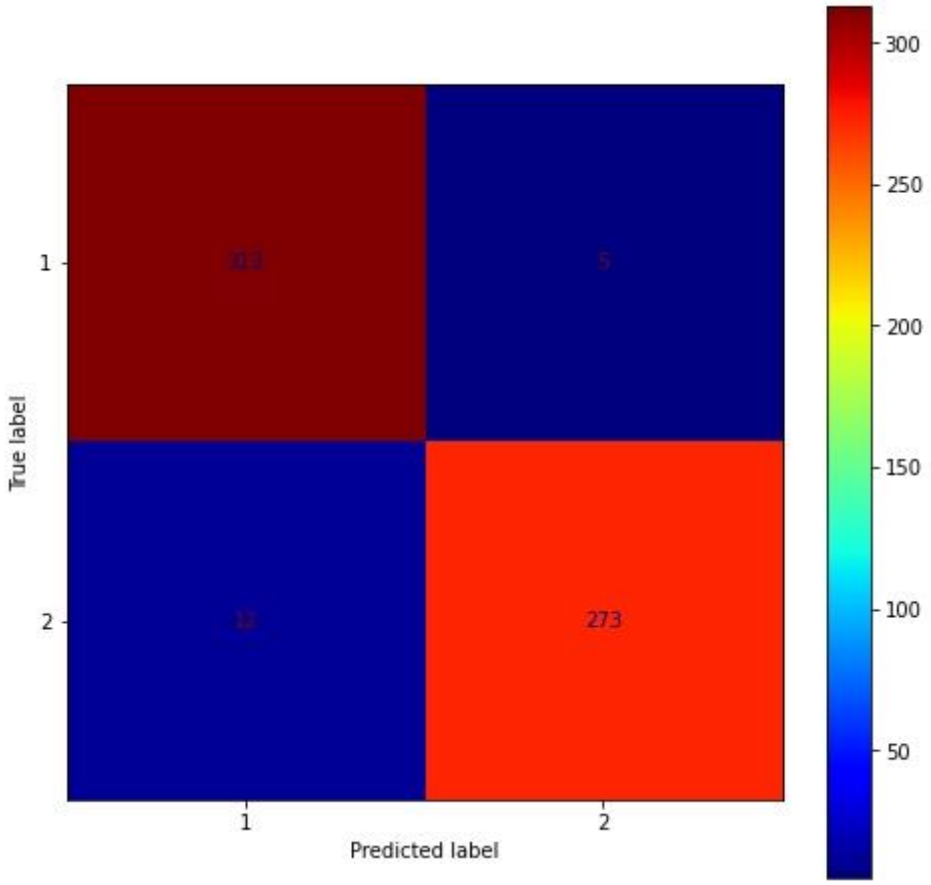

Figura 20 - Matrice di confusione ottenuta da un singolo test sull'Hockey Fight Dataset

Come già visto più volte, la classe 1 rappresenta l'aggressione, mentre la classe 2 rappresenta comportamenti normali. Considerando come classe positiva l'aggressione, nella diagonale principale abbiamo i risultati corretti, mentre in quella secondaria abbiamo gli errori. Partendo dal basso quindi, nella diagonale secondaria, abbiamo rispettivamente l'errore di tipo 1 e l'errore di tipo 2. In questo singolo test l'accuratezza registrata è del 97,18%.

Nell’immagine sottostante, invece, è raffigurata la curva AUC prodotta tramite 5-fold cross validation.

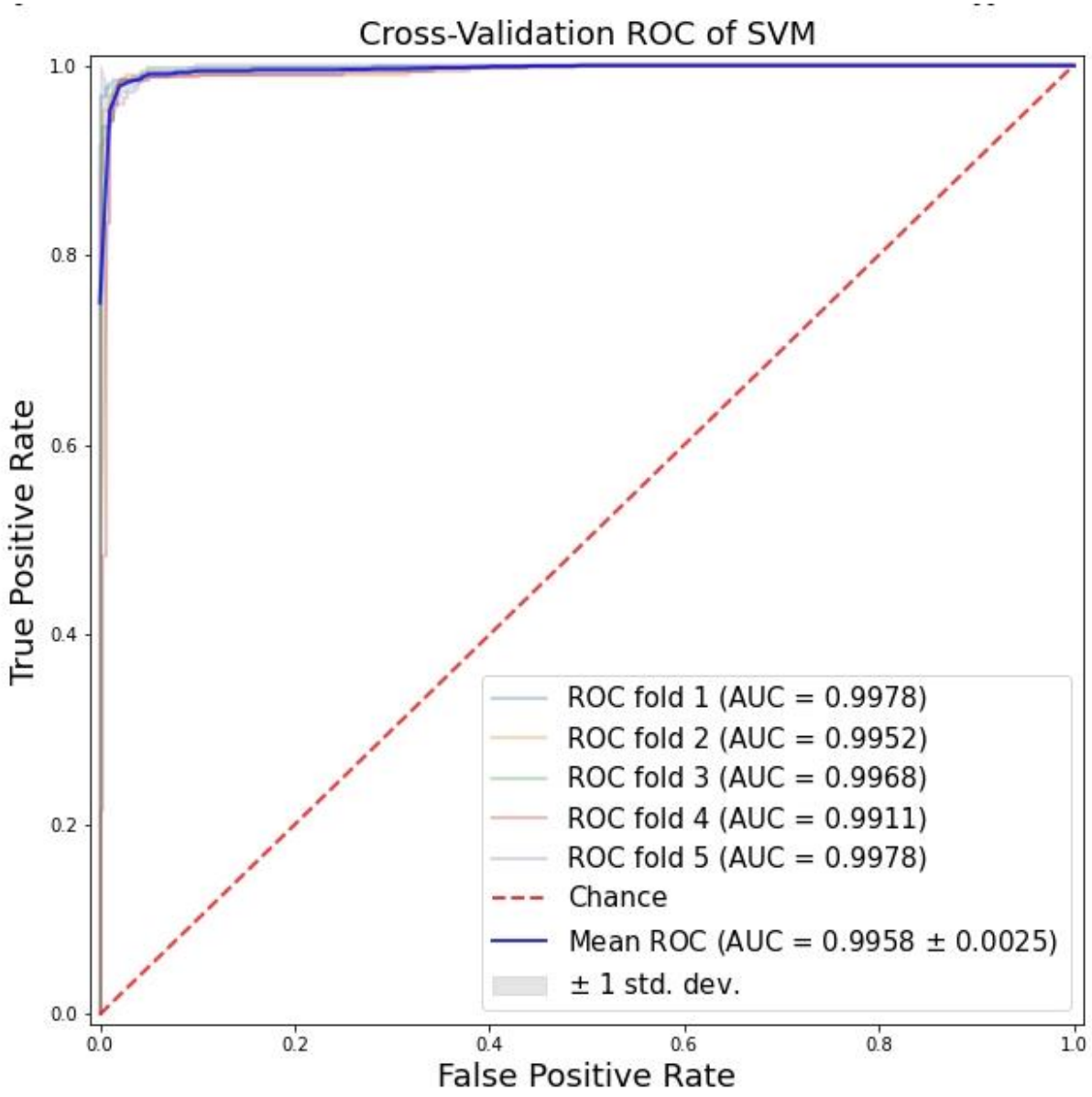

Figura 21 - Area Under Curve dell'Hockey Fight Dataset

5.3 – Test Crowd Violence Dataset

Come si ricava facilmente dalla tabella sottostante, nei casi di affollamento il sistema realizzato ottiene delle performance migliori rispetto agli approcci definiti nello stato dell’arte e praticamente identici a quelli di Accattoli.

Existing	Algorithm	ACC ± SD	AUC
	LHOG + LHOF + BoW	94,31 ± 1,65%	0,9703
	CNN + LSTM	94,57 ± 2,34%	-
	Two-stream + IDT	92,5 %	-
	MoSIFT + KDE + SC	89,05 ± 3,26 %	0,9357
	DIMOLIF	85,83 ± 4,26 %	0,8925
	Accattoli's C3D	**99,29 ± 0,59%**	**0,9900**
Proposed	**C3D**	**99,47 ± 0,67%**	**0.9998**

Tabella 2 - Confronto delle performance di classificazione nel Crowd Violence Dataset

Da questa tabella e dalla precedente si può notare una cosa che era già stata anticipata quando avevamo introdotto i vari dataset di benchmark, ovvero il fatto che tecniche che hanno alte performance sull'Hockey Fight Dataset, tendono ad avere un calo, in certi casi anche notevole, delle performance in situazioni affollate. Come era stato messo in luce anche da Accattoli, il Crowd Violence Dataset risulta essere una base di dati molto competitiva per via di queste caratteristiche:

- Pochi campioni;
- Scarsa risoluzione dei filmati;
- Presenza di molte persone che tendono a sovrapporsi risultando difficile l'estrazione di feature discriminanti.

Inoltre, il sistema ha il grande vantaggio di produrre pochi falsi negativi, che sono il caso di errore peggiore. Ad esempio, come era accaduto anche ad Accattoli, nel test relativo alla matrice di confusione rappresentata nella pagina seguente i casi negativi vengono predetti correttamente, infatti si è manifestato un unico errore di questo tipo. Oltretutto, anche i falsi positivi hanno ottenuto un ottimo risultato visto che le label sono state tutte predette correttamente non generando nessun errore. Comunque, in questo test l'accuratezza ottenuta è del 99,74%.

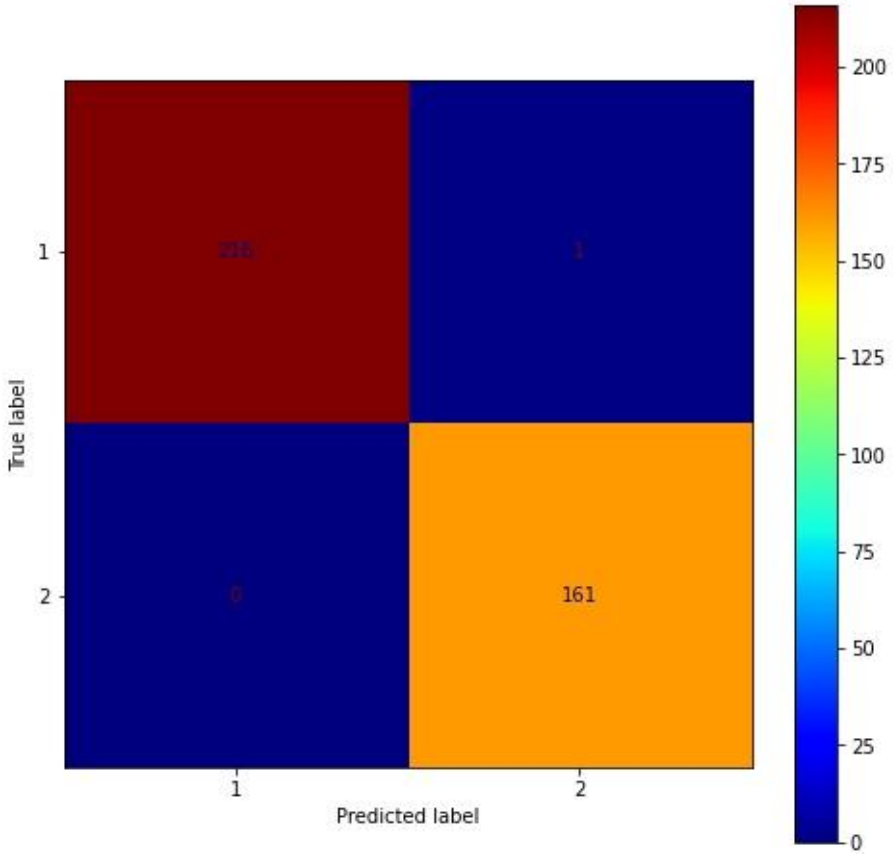

Figura 22 - Matrice di confusione ottenuta da un singolo test sul Crowd Violence Dataset

A differenza dell'Hockey Fight Dataset, questo dataset risulta essere di dimensione minore per via del numero limitato di video, che sono solo 246. Inoltre, non è perfettamente bilanciato per via della lunghezza variabile dei filmati.

Nella pagina seguente viene rappresentata la curva AUC prodotta tramite 5-fold cross validation.

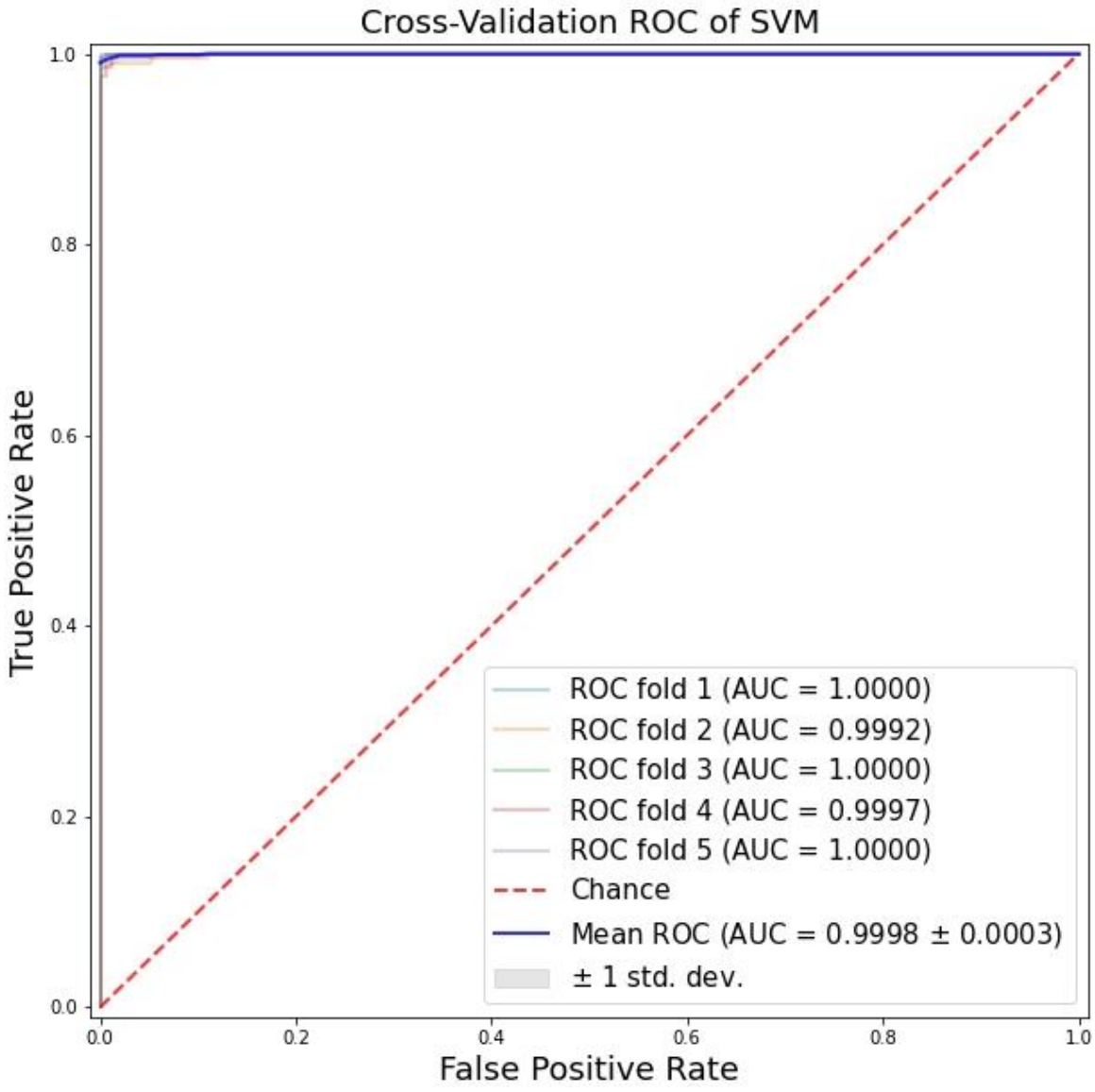

Figura 23 - Area Under Curve del Crowd Violence Dataset

5.4 – Test AIRTLab Violence Dataset

Oltre ai tre dataset utilizzati spesso in letteratura per la *violence detection*, come già accennato nel capitolo precedente, abbiamo deciso di testare il nostro sistema anche su un ulteriore dataset realizzato all'interno dell'AIRTLab dell'Università Politecnica delle Marche. Ricordiamo che questo dataset non è bilanciato, infatti abbiamo 120 video raffiguranti comportamenti normali e 230 video raffiguranti scene di violenza, oltre al fatto che, come nel caso del Crowd Violence Dataset, i filmati hanno lunghezza variabile.

Inoltre, come avevamo sottolineato nella breve descrizione di questo dataset, i video non violenti contengono spesso comportamenti (ad esempio abbracci, pacche sulla spalla, ecc…) che possono

causare dei falsi positivi, a causa dei movimenti veloci e delle somiglianze con alcuni comportamenti violenti. Questo porta ad ottenere un'accuratezza media minore rispetto ai precedenti test, infatti questa è del 95,42% con una deviazione standard pari all'1,04%, ottenuta come sempre tramite 5-fold cross validation. Oltretutto, questa caratteristica del dataset dell'AIRTLab porta anche a risultati peggiori nella matrice di confusione e nella curva AUC, proprio per la presenza generalmente di un numero superiore di falsi positivi rispetto a quelli che si verificavano nei due dataset analizzati precedentemente.

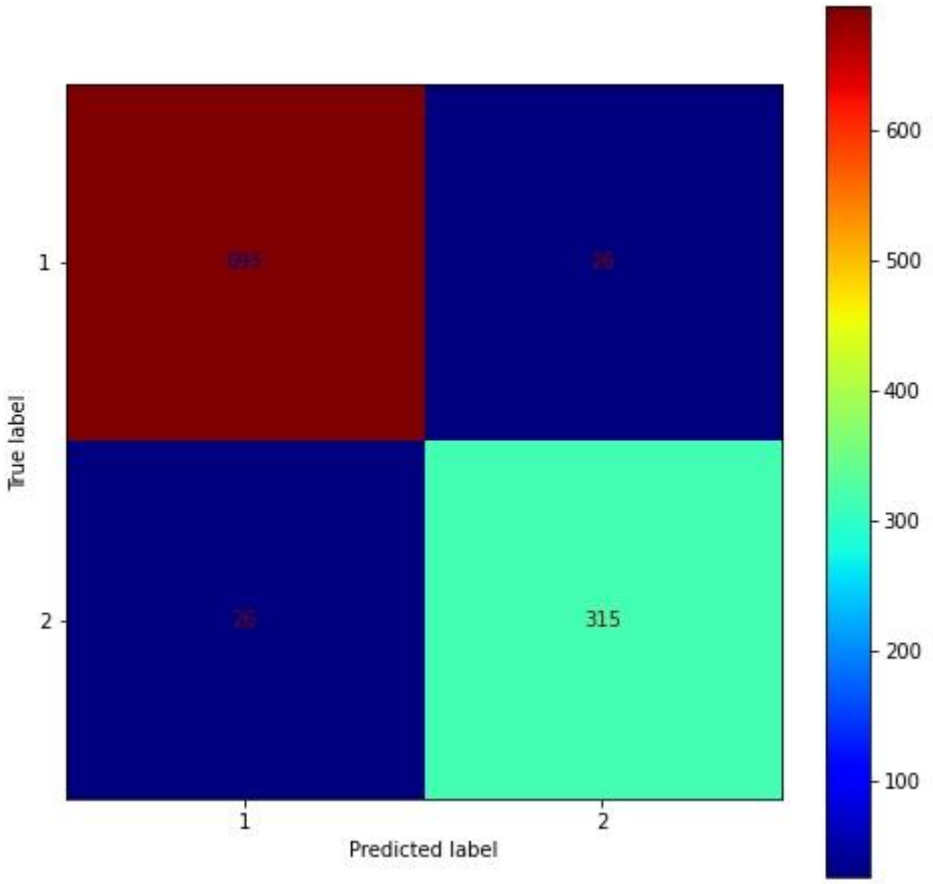

Figura 24 - Matrice di confusione ottenuta da un singolo test sull'AIRTLab Violence Dataset

Proprio in questo esempio, infatti, possiamo notare come il numero di errori del tipo 1 sia decisamente superiore a quello ottenuto nei test effettuati sui dataset precedenti. È poi interessante notare come in questo particolare esempio il numero dei falsi positivi e dei falsi negativi si eguaglino. In questo singolo test l'accuratezza registrata è del 95,10%.

Nell'immagine sottostante, invece, è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation. Come possiamo vedere, anch'essa conferma ciò che stavamo affermando, ottenendo risultati peggiori di quella ricavata per i due dataset precedenti.

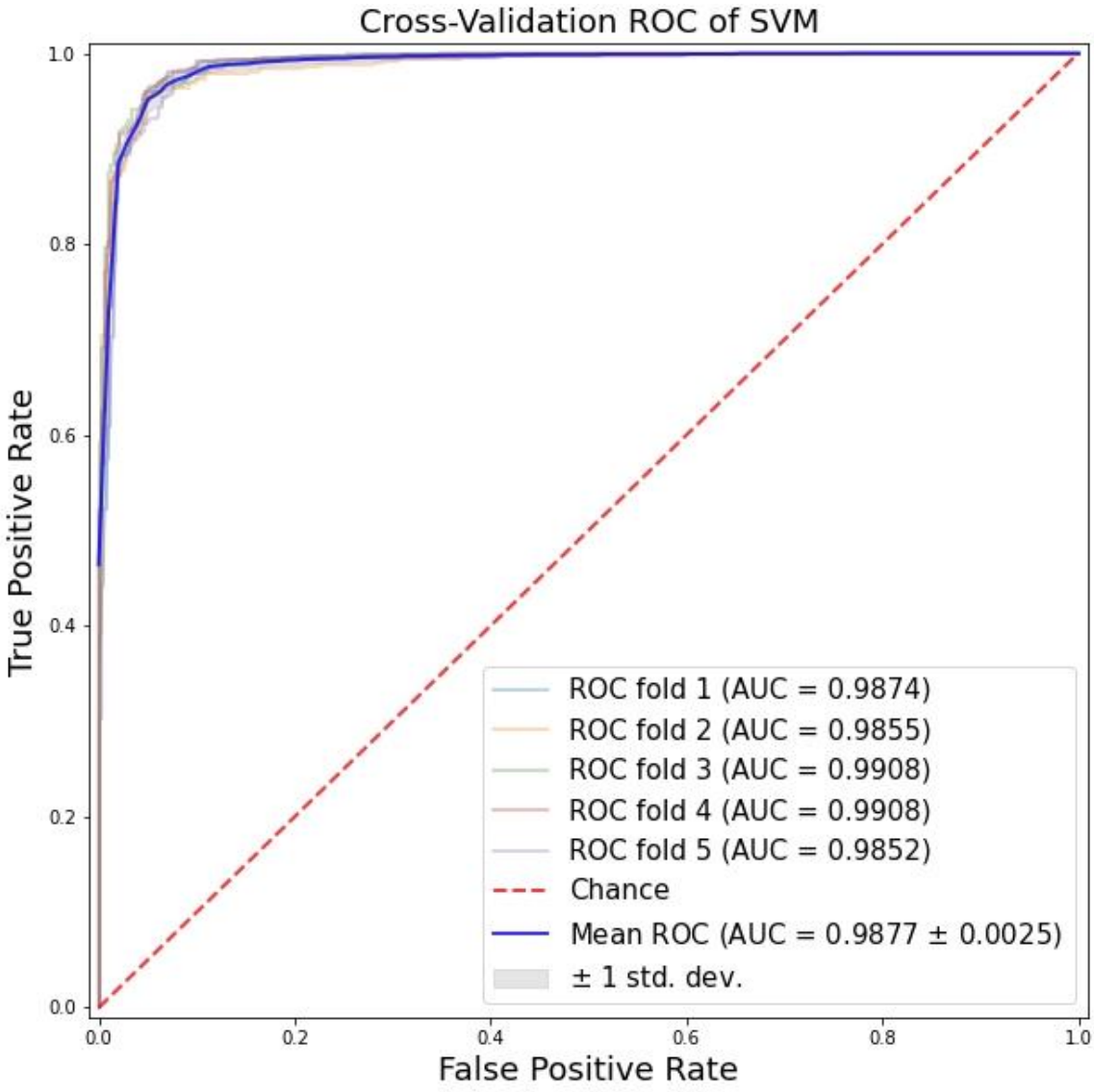

Figura 25 - Area Under Curve dell'AIRTLab Violence Dataset

5.5 – Test Dataset Intero

Per effettuare l'addestramento vero e proprio del classificatore SVM, come già illustrato nello scorso capitolo, e valutare la capacità di generalizzazione del sistema realizzato abbiamo utilizzato un'ulteriore base di dati formata dai tre dataset di benchmark famosi in letteratura per la *violence detection* che ormai conosciamo perfettamente, ossia Hockey Fight Dataset, Crowd Violence Dataset e Movie Violence Dataset. Utilizzare il dataset intero, inoltre, ci è servito anche a giudicare se è possibile usare il sistema in condizioni reali. Però, il problema centrale del riconoscimento in situazioni reali è quello di non essere in grado di esprimere in maniera esaustiva il caso del comportamento normale.

A differenza di quanto aveva fatto Accattoli, per questo test abbiamo deciso di usare la cross validation e di estrarre tutti i segmenti dai video dei tre dataset di benchmark, anche per il fatto che questa base di dati ci serviva per effettuare al meglio l'addestramento del classificatore secondo il principio espresso nel Capitolo 4, ossia scegliere come modello tra i vari classificatori addestrati con la tecnica della 5-fold cross validation quello che aveva ottenuto l'accuratezza maggiore. Comunque, tramite la cross validation su questo dataset abbiamo ottenuto un'accuratezza media del 98,18% con una deviazione standard dell'1,20%. Invece, effettuando un singolo test con il solito campionamento casuale, in cui usiamo il 70% dei dati come training set e il restante 30% per il test set, abbiamo ottenuto un'accuratezza del 98,17% e la seguente matrice di confusione.

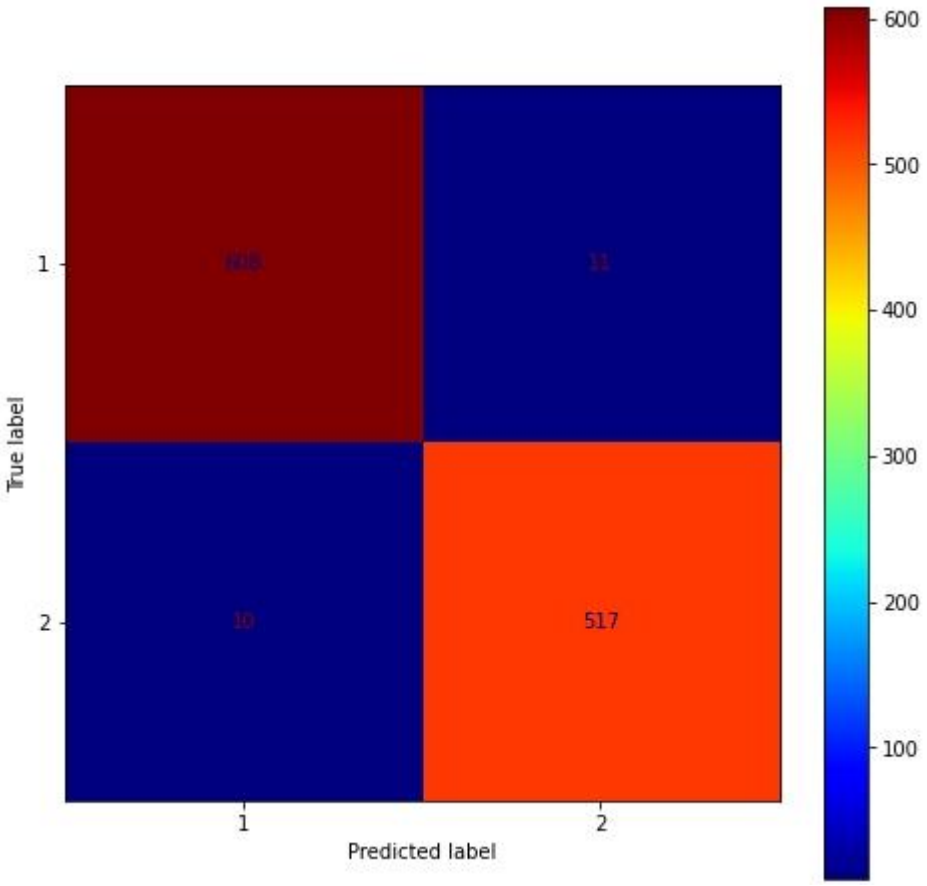

Figura 26 - Matrice di confusione ottenuta da un singolo test sul dataset intero

Nella pagina seguente riportiamo come al solito la curva AUC prodotta tramite 5-fold cross validation.

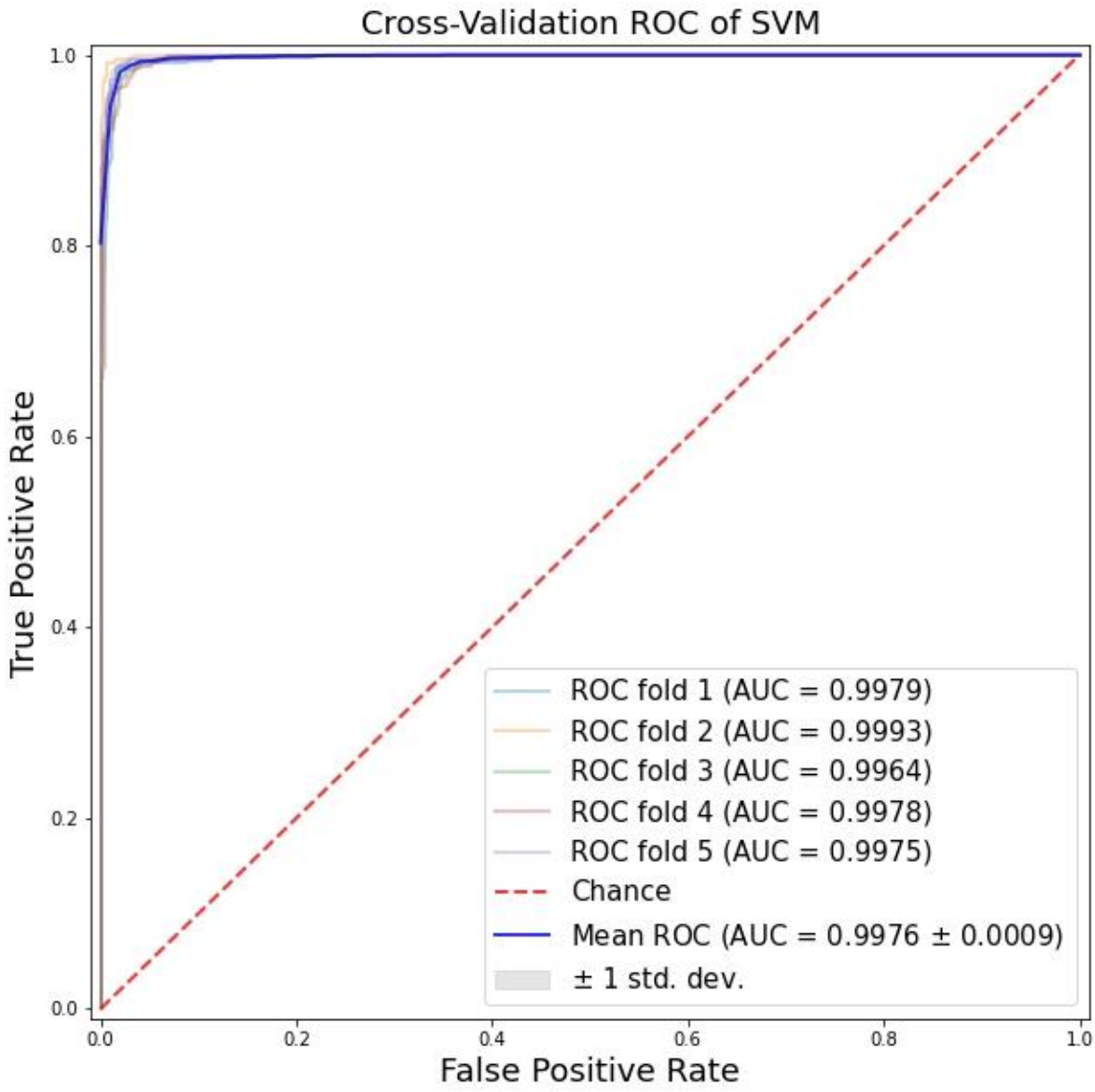

Figura 27 - Area Under Curve del dataset intero

5.6 – Analisi dei risultati ottenuti

Come possiamo vedere nelle due tabelle dell'Hockey Fight Dataset e del Crowd Violence Dataset, i risultati ottenuti confermano quelli già ottenuti precedentemente da Accattoli e quindi il nostro porting ha saputo replicare perfettamente gli esperimenti del progetto su cui è basato. L'analisi dei dati, infatti, anche in questo caso, mostra come il metodo utilizzato ottiene alti valori di accuratezza in entrambi i dataset di benchmark. Oltretutto si ricavano ottimi risultati anche sul dataset intero e sull'AIRTLab Violence Dataset. Questo conferma quindi come questo approccio abbia una migliore capacità di generalizzazione rispetto ad altre tecniche dello stato dell'arte, rendendolo quindi più versatile e utilizzabile in diverse circostanze.

Inoltre, un'altra conferma dei risultati ottenuti da Accattoli riguarda l'AUC che anche in questo caso possiamo notare come rimane molto simile ai valori di accuracy, tranne per l'AIRTLab Violence Dataset. Ma questo come spiegato precedentemente è assolutamente comprensibile considerando il fatto che questo dataset non è bilanciato e che i video non violenti presentano spesso comportamenti che possono causare falsi positivi. Comunque, questo risultato riguardante l'Area Under Curve nei restanti dataset come era stato sottolineato anche da Accattoli è dovuto a vari fattori:

- Ogni dataset è bilanciato;
- L'accuratezza alta tende ad assottigliarsi con l'AUC;
- Il numero di falsi positivi e falsi negativi risulta essere simile.

Un'altra conferma interessante della tesi precedente risiede negli errori di classificazione prodotti. L'analisi ha messo in luce anche in questo caso che:

- Alcuni dei falsi positivi erano caratterizzati da comportamenti amichevoli come abbracci o pacche sulle spalle, ad esempio, che risultano essere simili ad atti violenti, ma con un'intensità del contatto tra gli individui differente. Questo avviene in particolare nell'AIRTLab Violence Dataset per le sue caratteristiche sottolineate più volte.
- Numerosi falsi positivi sono dovuti al fatto che spesso nei 16 frame campionati non ci sia traccia di atti violenti. Questo, come accennato nel capitolo precedente, è dovuto al fatto che i filmati etichettati come violenti in alcuni casi contengono all'interno porzioni di frame che non contengono effettivamente un'aggressione.

Questo ci porta a concludere, come Accattoli, non solo che la rete ha un'accuratezza molto alta, ma che in determinate situazioni l'errore prodotto è dovuto ad input ambigui.

Comunque, confrontando anche in questo caso il nostro approccio con quello presentato in [14], che utilizza anch'esso le reti neurali 3D, possiamo affermare senza alcun dubbio che avere una rete neurale addestrata su un dataset differente e molto più ampio di quello di lavoro, produce un modello con una capacità di generalizzare migliore e delle feature più rappresentative. Inoltre, da questo confronto si possono esaminare le performance di un classificatore SVM rispetto ad una MLP. Infatti, i risultati mostrano che la SVM, oltre ad avere una velocità di classificazione superiore alla sua avversaria, risulta anche più performante per il problema della *violence detection*.

In conclusione, possiamo affermare che i risultati ottenuti confermano pienamente quelli ottenuti da Accattoli nel suo lavoro di tesi, ovvero che le tecniche basate sul deep learning hanno ottime performance per il riconoscimento automatico di scene di violenza.

Capitolo 6

Conclusioni e sviluppi futuri

Questo lavoro ha effettuato un porting utilizzando tecnologie attuali dell'architettura che era stata realizzata nel lavoro di tesi di Accattoli [1], che aveva usato il modello di rete C3D, una 3D Convolutional Neural Network che permette l'estrazione di feature riguardanti il movimento, per calcolare i descrittori dei video; descrittori che poi sono stati utilizzati come input per una SVM lineare affinché eseguisse la classificazione dei filmati in violenti o meno.

La tecnica utilizzata ha ottenuto gli stessi risultati di quella di Accattoli confermando di avere migliori performance delle altre tecnologie presenti nello stato dell'arte, in entrambi i casi di violenza considerati, sia in situazioni affollate che tra singoli individui. Quindi si può affermare senza alcun dubbio che il sistema realizzato ha soddisfatto pienamente gli obbiettivi che ci eravamo prefissati.

Purtroppo, però, le conferme dei risultati ottenuti da Accattoli le abbiamo anche per quanto riguarda i problemi che affliggono il nostro sistema. Questo, infatti, come abbiamo potuto vedere dall'analisi dei risultati, presenta dei falsi positivi nelle situazioni in cui sono rappresentati comportamenti amichevoli che sono simili ad atti violenti, come ad esempio gli abbracci o le pacche sulle spalle. Questo, come era stato sottolineato anche da Accattoli, potrebbe essere migliorato andando a considerare l'intensità del movimento, combinando le feature estratte dalla rete con l'informazione riguardante l'accelerazione. Inoltre, risulta quasi impossibile rappresentare la classe di "non aggressione", infatti, mentre la violenza ha un comportamento ben specifico, il caso pacifico è rappresentato da un concetto troppo generico. Per poter migliorare questo aspetto si potrebbe fornire in input più esempi di comportamenti normali, ma questo potrebbe aumentare il numero di falsi negativi, che sono errori molto peggiori dei falsi positivi. Un'altra soluzione potrebbe essere anche quella di prendere una decisione osservando la sequenza temporale dei rilevamenti prodotti dal sistema, pesando i casi positivi nel tempo.

Concludendo, avendo ottenuto gli stessi risultati di Accattoli, possiamo senza alcun dubbio affermare che il sistema realizzato è un porting perfetto dell'architettura del precedente lavoro di tesi e ha rispettato pienamente gli obiettivi prefissati.

Sono diversi gli sviluppi futuri che potrebbero essere fatti a partire dai risultati ottenuti in questo elaborato. Di seguito procediamo ad elencarne alcuni interessanti:

- Introdurre informazioni aggiuntive (come l'accelerazione) per migliorare l'accuratezza e la capacità discriminativa.
- Aggiungere diverse categorie di comportamenti violenti, implementando quindi un classificatore multi-classe.
- Testare il sistema in condizioni reali, introducendo un addestramento continuo, il cosiddetto continuos learning, per cercare di apprendere in modo sempre più completo il comportamento normale.
- Ricercare e testare altri modelli di reti neurali da sostituire a C3D per vedere se esistono reti in grado di ottenere performance ancora migliori del sistema che abbiamo realizzato. Un possibile sviluppo interessante, in merito a questo punto, potrebbe essere quello di sostituire il modello che abbiamo implementato, costituito da C3D e il classificatore SVM, con una rete end-to-end. Con questa soluzione al posto di dover mettere in cascata un classificatore alla nostra rete, si affronterebbe il problema della *violence detection* con un'unica rete neurale riaddestrando solamente i layer finali.

Bibliografia

[1] S. Accattoli, "*Riconoscimento in tempo reale di scene di violenza utilizzando il Deep Learning*", Tesi di Laurea Magistrale, UNIVPM, A.A. 2017/18, Rel. Aldo Franco Dragoni.

[2] [Online] Wikipedia: https://it.wikipedia.org/wiki/Violenza [Consultata il giorno 29 novembre 2020].

[3] D. Tran, L. Bourdev, R. Fergus, L. Torresani e M. Paluri, "*Learning spatiotemporal features with 3d convolutional networks*", ICCV, pp. 4489-4497, Dicembre 2015.

[4] S. J. W. Xu, M. Yang e K. Yu, "*3D Convolutional Neural Networks for Human Action Recognition*", IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 35, n. 1, pp. 221-231, 2013.

[5] M. Yang, S. Ji, W. Xu, J. Wang, F. Lv, K. Yu, Y. Gong, M. Dikmen, D. Lin e T. Huang, "*Detecting Human Actions in Surveillance Videos*", TREC Video Retrieval Evaluation Workshop, 2009.

[6] S. Ji, W. Xu, M. Yang e K. Yu, "*3D Convolutional Neural Networks for Human Action Recognition*", 27th Int'l Conf. Machine Learning, pp. 495-502, 2010.

[7] G. Taylor, R. Fergus, Y. LeCun e A. C. Bregler, "*Convolutional Learning of SpatioTemporal Features*", 11th European Conf. Computer Vision, pp. 140-153, 2010.

[8] Z. Fang, F. Fei, Y. Fang, C. Lee, N. Xiong e L. Shu, "*Abnormal event detection in crowded scenes based on deep learning*", Multimedia Tools and Applications., vol. 75, n. 22, p. 14617–14639, 2016.

[9] P. Zhou, Q. Ding, H. Luo e X. Hou, "*Violent Interaction Detection in Video Based on Deep Learning*", Journal of Physics: Conference Series., 2017.

[10] D. Xu, E. Ricci, Y. Yan, J. Song e N. Sebe, "*Learning Deep Representations of Appearance and Motion for Anomalous Event Detection*", BMVC, 2015.

[11] Z. Dong, J. Qin e Y. Wang, "*Multi-stream Deep Networks for Person to Person Violence Detection in Videos*", Chinese Conference on Pattern Recognition. Springer, pp. 517-531, 2016.

[12] S. Sudhakaran e O. Lanz, "*Learning to Detect Violent Videos using Convolutional Long Short-Term Memory*", 2017.

[13] Z. Meng, J. Yuan e Z. Li, "*Trajectory-Pooled Deep Convolutional Networks for Violence Detection in Videos*", Liu M., Chen H., Vincze M. (eds) Computer Vision Systems. Lecture Notes in Computer Science, ICVS, vol. 10528, 2017.

[14] C. Ding, S. Fan, M. Zhu, W. Feng e B. Jia, "*Violence Detection in Video by Using 3D Convolutional Neural Networks*", Bebis G. et al. (eds) Advances in Visual Computing. ISVC 2014. Lecture Notes in Computer Science, vol. 8888, 2014.

[15] B. George e K. Sangho, "*Spatio-temporal Networks: Modeling and Algorithms*", Springer-Verlag, New York, 2013.

[16] T. Mikolov, "*Recurrent neural network based language*", Brno University of Technology, Johns Hopkins University, 2010.

[17] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar e F.-F. L, "*LargeScale Video Classification with Convolutional Neural Networks*", 2014, IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH,, pp. 1725-1732, 2014.

[18] C. Corinna e V. Vapnik, "*Support-vector networks*", Machine learning 20.3, pp. 273-297, 1995.

[19] Y. Bengio, "*Learning Deep Architectures for AI*", Foundations and Trends® in Machine Learning, vol. 2, n. 1, pp. 1-127, 2009.

[20] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach e J. Martens, "*Adding gradient noise improves learning for very deep networks*", 2015.

[21] I. Goodfellow, Y. Bengio e A. Courville, "*Deep Learning*", 2015.

[22] University of central Florida, "*UCF - Center for Research in Computer Vision*", 2012. [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.php [Consultato il giorno 1 Dicembre 2020]

[23] K. Simonyan e A. Zisserman, "*Very deep convolutional networks for large scale image recognition*", conference paper at ICLR, 2015

[24] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga e G. Toderici, "*Beyond short snippets: Deep networks for video classification*", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4694-4702, 2015.

[25] I. Misra, C. L. Zitnick e M. Hebert, "*Shuffle and learn: unsupervised learning using temporal order verification*", ECCV, 2016.

[26] K. Simonyan e A. Zisserman, "*Two-stream convolutional networks for action recognition in videos*", NIPS, 2014.

[27] [Online] Google Colab: https://colab.research.google.com/notebooks/intro.ipynb

[28] [Online] Jupyter: https://jupyter.org

[29] [Online] Keras: https://keras.io

[30] [Online] Sklearn: https://scikit-learn.org/stable

[31] [Online] OpenCV: https://opencv.org

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama e T. Darrell, "*Caffe: Convolutional Architecture for Fast Feature Embedding*", arXiv:1408.5093, 2014.

[33] E. Bermejo Nievas, O. Deniz Suarez, G. Bueno García e R. Sukthankar, "*Violence Detection in Video Using Computer Vision Techniques*", Real P., Diaz-Pernil D., Molina-Abril H., Berciano A., Kropatsch W. (eds) Computer Analysis of Images and Pattern CAIP 2011. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, vol. 6855, 2011.

[34] T. Hassner, Y. Itcher e O. Kliper-Gross, "*Violent flows: Real-time detection of violent crowd behavior*", Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2012.

[35] [Online] Available: https://github.com/airtlab/A-Dataset-for-Automatic-Violence-Detection-in-Videos

[36] [Online] Available: https://github.com/aslucki/C3D_Sport1M_keras