UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica e dell' Automazione

Esperimenti di Machine Learning e Deep Learning nell'ambito del riconoscimento automatico di scene di violenza in flussi video

Machine Learning and Deep Learning experiments in the field of automatic recognition of violence scenes in video streams

Relatore:
Prof. Aldo Franco Dragoni

Tesi di Laurea di:
Erminio Ciuffreda

Correlatore:
Dott. Paolo Sernani

A.A. 2020/2021

Ringraziamenti

Grazie alla mia famiglia, che mi ha sempre incoraggiato in questi anni, sostenendomi nelle scelte e permettendomi di arrivare a questo traguardo.

Grazie ai professori Dragoni e Sernani per avermi dato la possibilità di questo lavoro di tirocinio, ed essermi stati di aiuto durante questo periodo.

Grazie ai miei compagni di corso ed amici che mi hanno aiutato nei momenti del bisogno e aver reso ogni momento bello ed indimenticabile.

Ancona, Luglio 2021

Erminio Ciuffreda

Abstract

Con l'emergere del concetto di "città sicura", la costruzione della sicurezza è stata gradualmente apprezzata da varie città e anche la tecnologia di videosorveglianza è stata continuamente sviluppata e applicata. Tuttavia, poiché i requisiti funzionali delle applicazioni reali diventano sempre più diversificati, anche i sistemi di videosorveglianza devono essere più intelligenti. Avere un sistema automatico capace di eseguire il monitoraggio continuo delle immagini, permetterebbe di migliorare notevolmente l'efficienza della video sorveglianza.

In questo elaborato viene introdotto ed implementato un sistema automatico per la scelta degli iperparametri migliori da utilizzare all'interno di un sistema di riconoscimento che, utilizzando due architetture basate sulle 3D Convolutional Neural Network, è in grado di rilevare lotte, movimenti aggressivi e scene di violenza nei video. Le due architetture vengono confrontate utilizzando sette dataset di benchmark, tre dei quali sono quelli più famosi in letteratura (Hockey Fights dataset, Crowd Violence dataset e Movie Violence dataset).

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1.

Introduzione

Con il termine "violenza" intendiamo ogni atto o comportamento che faccia uso della forza fisica (con o senza l'impiego di armi o di altri mezzi di offesa) per recare danno ad altri nella persona o nei suoi beni o diritti. In senso più ampio, l'abuso della forza (rappresentata anche da sole parole, o da sevizie morali, minacce, ricatti), come mezzo di costrizione, di oppressione, per obbligare cioè altri ad agire o a cedere contro la propria volontà [5].
Nella società odierna la violenza è sempre più diffusa, pur disponendo di sistemi di sorveglianza sempre più all'avanguardia il tasso di criminalità in alcuni paesi rimane molto elevato. Il lavoro di tesi qui svolto si basa sull'interpretazione della violenza come un'azione volontaria attuata come costrizione materiale.

La *Violence Detection* si pone come obiettivo il riconoscere in maniera automatica una aggressione tra due o più persone. Una delle due architetture proposte è una soluzione già adottata in un lavoro di tesi precedente di uno studente dell'Università Politecnica delle Marche, Simone Accattoli [1], che utilizza una rete neurale convoluzionale 3D per rilevare il verificarsi di un atto violento all'interno di una scena visiva ripresa da un sistema di sorveglianza.

1.1. Motivazioni

La videosorveglianza è sempre stata uno degli strumenti più utili per permettere di rafforzare la protezione delle persone, sia in ambienti pubblici che privati, visto che permettono di monitorare comportamenti anormali ed indesiderati. Solitamente il monitoraggio viene eseguito 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, anche a causa dei numerosi monitor da controllare.

La *violence detection* parte dalle tecniche utilizzate per la *human action recognition* per il riconoscimento di una particolare azione o comportamento che classifichiamo come violento. Abbiamo deciso di utilizzare un modello di rete convoluzionale 3D pre-addestrato chiamato C3D [4] 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), un classificatore non lineare molto utilizzato in letteratura.

Questo lavoro, nasce come continuazione di una lavoro precedentemente eseguito da un altro studente dell'Università Politecnica delle Marche, Simone Cappanera [6], che ha effettuato un porting dell'architettura realizzata da Accattoli, eseguendo nuove modifiche così da ottimizzare il codice e migliorare le sue prestazioni.
Oltre all'architettura utilizzata con SVM, si è andato a testare un modello con architettura end-to-end alla ricerca dei migliori *hyperparameter*.

I campi applicativi sono numerosi e a titolo di esempio ne riportiamo alcuni qui di seguito:

- *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 il Grind search per l'automatizzazione nella ricerca della miglior combinazione di Hyperparameter basato sul porting dell'architettura effettuato da Cappanera.
- Valutare i migliori Optimizer nell'architettura end-to-end.
- Testare dataset differenti da quelli già presenti nello stato dell'arte utilizzati per la *violence detection*, e valutare i risultati ottenuti.

1.3. Struttura della tesi

La presentazione del lavoro svolto è suddiviso nel modo seguente:

- Nel secondo capitolo introdurremo velocemente lo stato dell'arte riguardante l'argomento trattato. In particolare, approfondiremo le tecniche maggiormente utilizzate per quanto riguarda il *deep learning* e come vengono applicate 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 ad illustrare l'intero sistema utilizzato, indicando i vari strumenti impiegati, i vari dataset, l'architettura del sistema di riconoscimento, gli optimizer scelti per l'architettura end-to-end.
- 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; inoltre verranno svolti degli esperimenti anche su altri dataset: AIRTLab[1], Real Life Violence [7], Surveillance Camera Video [8].

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

- Infine, il sesto capitolo mostra l'analisi conclusiva del progetto con i possibili sviluppi futuri.

Capitolo 2.

Lo Stato dell'Arte

In questo capitolo affronteremo sinteticamente gli aspetti fondamentali della *violence detection*, andremo ad analizzare i concetti teorici e tecnologici presenti nello stato dell'arte, soffermandoci sulle tecniche basate sul *deep learning*.

2.1. Violence Detection

La *violence detection* si inserisce come un particolare caso di *human 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 fondamentli, le *informazioni spazio-temporali*:

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

Le tecniche utilizzate per la *violence detection* sono varie, un caso particolare sono quelle che si basano sul deep learning. Recenti studi [9] [10] [11] [12] mostrano come queste tecniche hanno raggiunto alte performance anche nel riconoscimento comportamentale.

2.2. Deep learning e 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* [13] [14] [15] [2] [16] [17] [3].

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 sono divisibili in due categorie:

- *Spatio-temporal Networks*. Come viene mostrato in [18] 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 [19]. 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. Le RNN, invece, sono delle reti neurali in cui vengono permesse delle connessioni a retroazione, introducendo così una memoria.

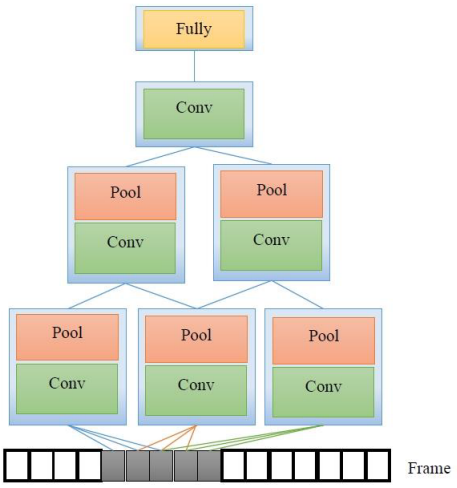

Figura 2.1.: Esempio di Spatio-temporal Network in modalità late-fusion tratto da [1]

- *Multiple Stream Networks.* Come illustrato nei lavori [2] [17], 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.

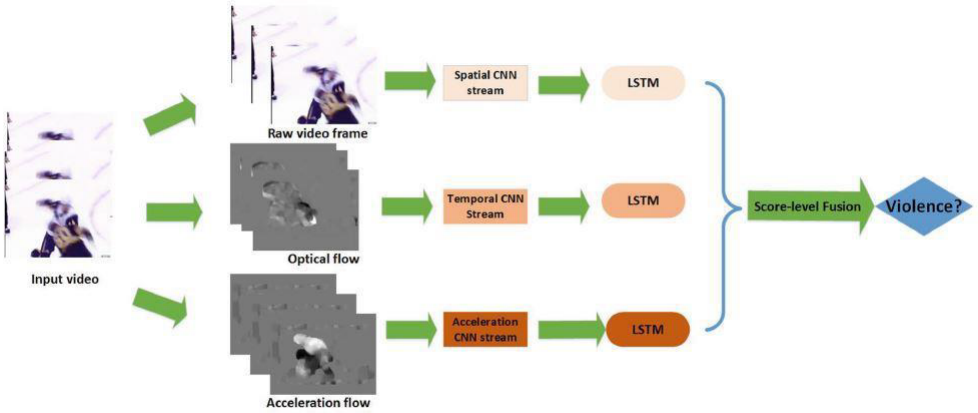

Figura 2.2.: Esempio di multi-stream neural networks tratto da [2]

2.2.1. Tecniche Spatio-Temporal

Ding et alii [3] propongono l'utilizzo di una rete convoluzionale tridimensionale per il problema della *violence detection*. Una 3D ConvNet non è altro che una rete convoluzionale standard in cui viene aggiunta la dimensione temporale. La rete illustrata in [3] 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.

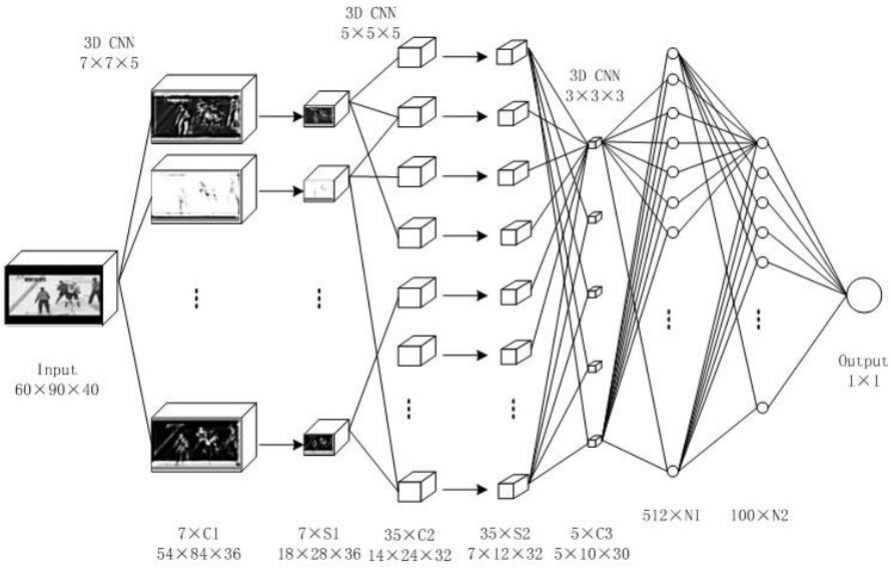

Figura 2.3.: Architettura rete proposta in [3]

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.
Le differenze sostanziali del modello preso in esame rispetto a quello di Ding et alii sono:

- L'architettura della rete. Nonostante entrambe le soluzioni siano delle 3D ConvNet le architetture risultano differenti.
- Il modello proposto 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 [20]. Esse non sono altro che dei modelli supervisionali di Machine Learning usati per la regressione o la classificazione.
L'SVM è basato sull'idea di trovare un iperpiano che divida al meglio un set di dati in due classi. Per comprenderne il funzionamento è bene definire alcuni concetti chiave:

- *Iperpiano* o limite di decisione lineare: in due dimensioni è raffigurato come una linea che separa e classifica un insieme di dati.

- *Support Vector*: essi sono i punti dati più vicini all'iperpiano. Tali punti dipendono dal set di dati che si sta analizzando e se vengono rimossi o modificati alterano la posizione dell'iperpiano divisorio.

- *Margine*: è definito come la distanza tra i vettori di supporto di due classi differenti più vicini all'iperpiano. Alla metà di questa distanza viene tracciato l'iperpiano, o retta nel caso si stia lavorando a due dimensioni.

Il Support Vector Machine ha l'obiettivo di identificare l'iperpiano che meglio divide i vettori di supporto in classi. Teoricamente, è possibile disegnare un numero infinito di linee rette. Il problema è trovare quale tra le infinite rette risulti ottimale, ossia quella che generi il minimo errore di classificazione su una nuova osservazione. Per spiegare il funzionamento di un classificatore basato sulle SVM riportiamo un esempio 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:

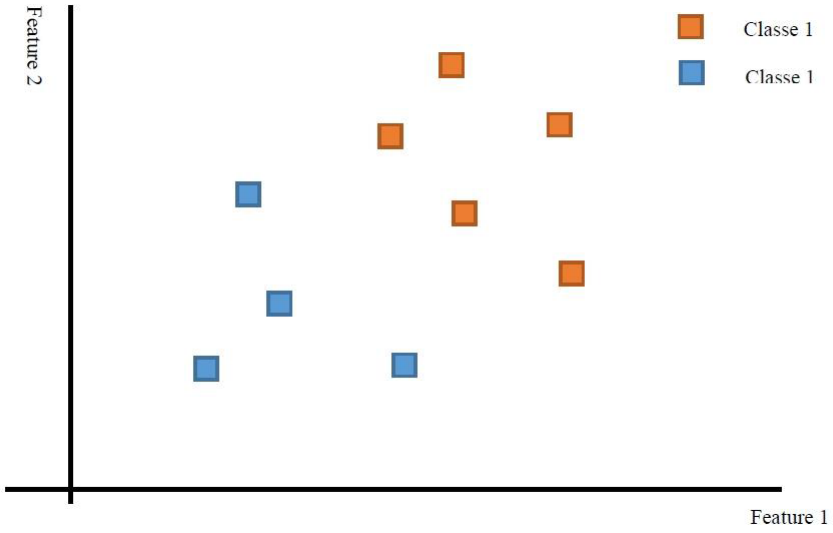

Figura 2.4.: Esempio di dataset di training tratto da [1]

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. Graficamente possiamo vederlo come segue:

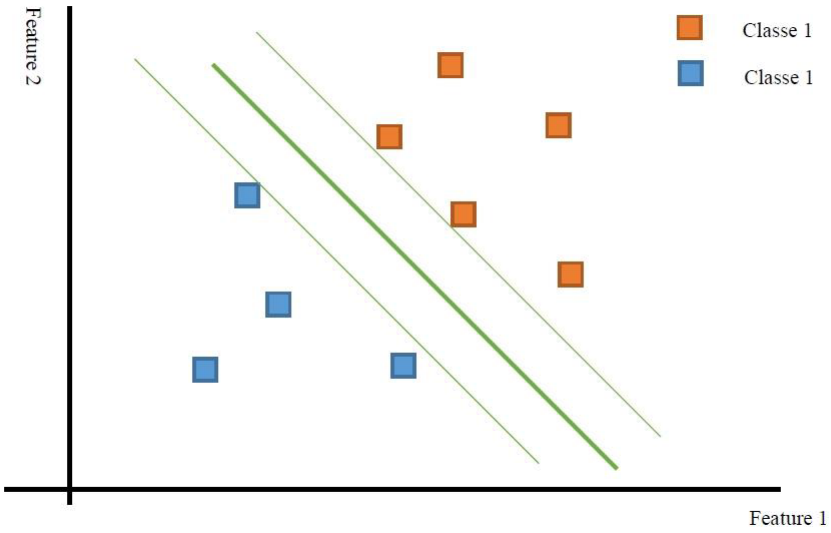

Figura 2.5.: Esempio di bordo di decisione tratto da [1]

In questa rappresentazione grafica, la retta verde più spessa rappresenta la linea di decisione lineare 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. Infatti, più lontano dalla linea di decisione si trovano i nostri punti dati, più siamo fiduciosi che essi siano stati classificati correttamente. Pertanto, desideriamo che i nostri punti dati siano il più lontano possibile dalla linea di decisione, pur rimanendo nella parte corretta. Quindi, quando vengono aggiunti nuovi dati di test, il modello decide la classe che assegniamo ad essa. Questo è ciò che fa il Support Vector Machine nella pratica, quando trova un iperpiano linearmente separabile.

Un uso alternativo per SVM è il metodo del kernel, che ci consente di modellare modelli non lineari di dimensioni superiori. Gli algoritmi SVM utilizzano un insieme di funzioni matematiche definite come kernel. Il suo scopo è di prendere i dati come input e trasformarli nella forma richiesta qualora non sia possibile determinare un iperpiano

linearmente separabile, come avviene nella maggior parte dei casi. Le funzioni Kernel più diffuse sono le seguenti:

- Kernel lineare.
- Kernel polinomiale.
- Kernel RBF, anche chiamato il kernel gaussiano.

2.4. Scelta del metodo utilizzato

Le motivazioni che ci hanno portato ad optare per le tecniche basate sul *deep learning* sono diverse. Innanzitutto 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. 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.
In particolare, tra le varie tecniche basate sul *deep learning* abbiamo 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 preprocessing, oltre al fatto che non si ha bisogno di informazioni a priori sul problema.

Capitolo 3.

Rete Neurale Convoluzionale C3D

La rete neurale convoluzionale 3D che è stata usata in questo lavoro di tesi 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 illustreremo nel prossimo capitolo, abbiamo deciso di utilizzare due modelli differenti:

- una combinazione di C3D come estrettatore di feature e una SVM per eseguire la classificazione dei video in violenti o meno;
- una combinazione di C3D come estrattore di feature e due fully connected layers per ottenere un network end-to-end per la classificazione.

In questo capitolo illustreremo velocemente i concetti alla base del funzionamento delle reti convoluzionali. La teoria introdotta in questo capitolo è stata ripresa dal libro "Deep Learning" [21].

3.1. Architettura delle reti convoluzionali

Convolutional Neural Network (CNN) è un nome che nasce dall'operazione matematica sfruttata, per l'appunto la Convoluzione, che è un tipo particolare di operazione lineare, come viene indicato in [21]. Le *CNN* sono progettate per riconoscere dei pattern visivi in modo diretto e non richiedono molto preprocessing o comunque ne richiedono una quantità molto limitata; rispetto alle normali reti neurali *Multi Layer Perception (MLP)*

abbiamo una riduzione di connessioni. Questo è dovuto all'introduzione di nuovi layer che sostituiscono quelli fully-connected delle MLP, chiamati così perchè tutti i neuroni di uno strato sono connessi con tutti i neuroni dello strato precedente. Le caratteristiche delle *CNN* che le differenziano dalle classiche *Multi Layer Perception* 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 un numero limitato 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 dei pesi da memorizzare.

- *Parameter sharing (Weight sharing).* Il valore dei pesi applicati ad un input è legato al valore dei pesi applicati dovunque. In una rete Convoluzionale oltretutto ogni membro del *kernel* è usato da tutti gli input e quindi la condivisione dei parametri offre il vantaggio di imparare un unico set di parametri piuttosto che un set di parametri distinto per ogni locazione.

I pesi di una *CNN* sono i parametri allenabili della rete, utilizzati per il processamente dell'input.

I layer che compongono una rete convoluzione sono i seguenti:

- Input Layer: primo layer che riceve l'immagine e la ridimensiona per poi passarla ai layer successivi.

- Layer Convoluzionale: ha lo scopo di estrarre le features.

- Layer di Pooling: I set di features ottenuti sono passati a questo strato che prende immagini di grandi dimensioni e le contrae preservando le informazioni fondamentali. Questo livello, inoltre, rende l'input delle features più piccolo e gestibile riducendo il numero di parametri sulla rete e di conseguenza ottimizzando la computazione.

- Layer Fully-connected: il suo scopo è quello di usare le features per classificare le immagini.

- Layer Softmax : funzione che permette la somma delle probabilità di un layer.

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

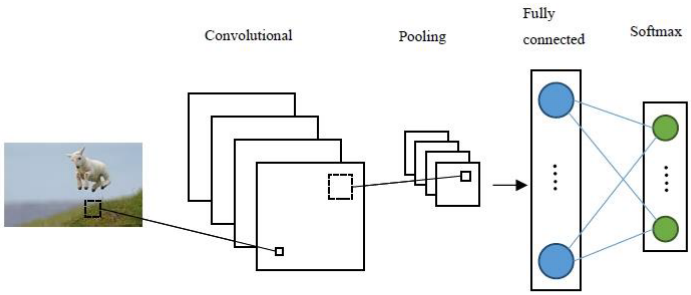

Figura 3.1.: Esempio architettura di una CNN tratto da [1]

3.1.1. Layer Convoluzionale

Questo è lo strato fondamentale che caratterizza una rete convoluzionale. Tale layer ha il compito 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 permette di preservare la relazione spaziale tra i pixel.

Il principio di funzionamento dello strato convoluzionale è quello di convolvere il kernel (definito sullo strato) su tutto l'input fornito alla rete. Questo produce un output (feature estratta) che può subire ulteriori elaborazioni dagli strati successivi per apprendere ed estrarre delle feature sempre di più alto livello.

Facendo un riassunto prettamente pratico ma anche semplicistico, si può dire che una CNN apprende i valori di questi filtri durante il training e si evince che quanti più filtri si useranno tante più features si estrarranno.

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

- *Depht:* il numero di filtri utilizzati per estrarre le feature map.
- *Stride*: numero di pixel con cui ci si sposta con la nostra matrice di filtro sulla matrice di input.
- *Zero-padding*: numero di zeri di padding usati per completare la matrice di input lungo i bordi usati.

In aggiunta un'altra caratteristica da fissare è la cosidetta *filter size*, cioè la dimensione dei filtri.

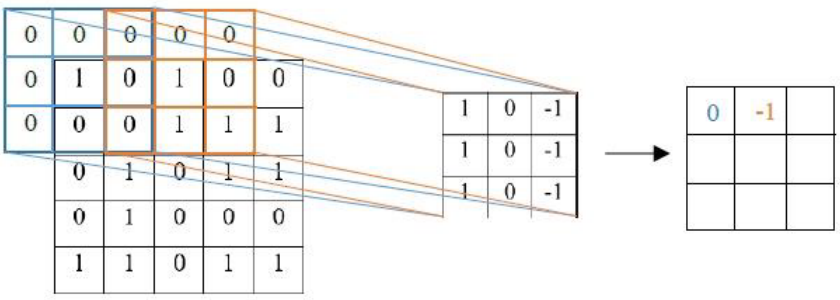

Figura 3.2.: Esempio di convoluzione con un singolo filtro, utilizzando lo zero-padding e con stride pari a 2 tratto da [1]

3.1.2. Layer di Pooling

Il layer di pooling serve a ridurre progressivamente la dimensione delle *feature map*, eliminando l'informazione superflua, ma mantendo quella utile. Questo viene fatto applicando una funzione alla *feature map* chiamata appunto operazione di pooling. Non esiste un solo tipo di operazione ma ne esistono differenti in base alla funzione applicata, tra le più note in letteratura abbiamo il Max Pooling o l'Average Pooling.
Facciamo un esempio concreto per spiegare meglio il funzionamento. Nel caso di Max Pooling si riprende il concetto del vicino spaziale prendendo il più grande elemento della mappa dentro la finestra scelta. Permette di identificare se la caratteristica di studio è presente nel livello precedente. Questo rende l'input delle features molto più piccolo e gestibile.

I vantaggi nell'utilizzo del pooling sono:

- Rendere le feature estratte invarianti alle trasformazioni affini come rotazioni, traslazioni, distorsioni, ecc...
- Limita l'overfitting
- Diminuire il numero dei parametri e il tempo di computazione tramite la riduzione della dimensione delle feature map.
- Generare una rappresentazione ridotta e più semplice da utilizzare.

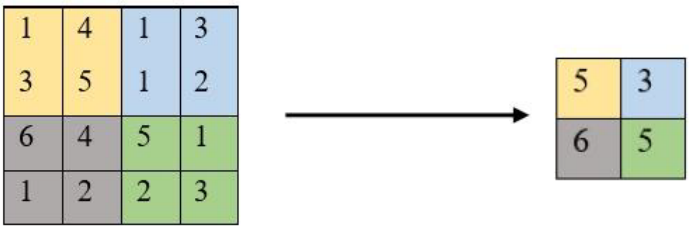

Figura 3.3.: Esempio di applicazione del Max Pooling con stride 2 e filtro di dimensione 2x2 tratto da [1]

3.1.3. Layer Fully-connected e Softmax

Questi layer sono solitamente posti alla fine della rete e si occupano di prendere le immagini filtrate ad alto livello e tradurre in categorie ciò che hanno analizzato. Il Layer *fully-connected (FC)* non è altro che un *Multi Layer Perceptron* che usa una funzione di attivazione Softmax, che permette di restituire un grado di affinità tra le classi utilizzate nell'addestramento della rete.
I due layer prendono in ingresso le feature di alto livello estratte dagli strati precedenti ed eseguono la classificazione dell'input. Per questo vengono posti in fondo alla rete.
L'unico parametro, che bisogna specificare, in questi layer è il numero di neuroni presenti nello strato. L'output prodotto sarà un vettore monodimensionale.

3.1.4. Training

Rimane ora da definire come viene utilizzato il tutto durante il training 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.

Nel caso del modello C3D combinato con SVM, l'algoritmo utilizzato è lo *stochastic gradient descent* con la *backward propagation* [21]. I vari passi del processo di addestramento sono riassunti qui di seguito:

- Si inizializzano i parametri dei kernel e dei pesi in maniera randomica.
- 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.
- 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.
- 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.

Nel modello C3D combinato con due fully-connected layer per ottenere l'architettura end-to-end, essendo l'algortimo di training uno dei Hyperparametres sotto studio, sono stati scelti 3 algoritmi, i quali verranno approfonditi nel capitolo successivo:

- Adam

- AdaDelta
- RMSProp

3.2. Reti convoluzionali 3D

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. Le *Convolutional Neural Network*, come viene indicato in [4], 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, 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.

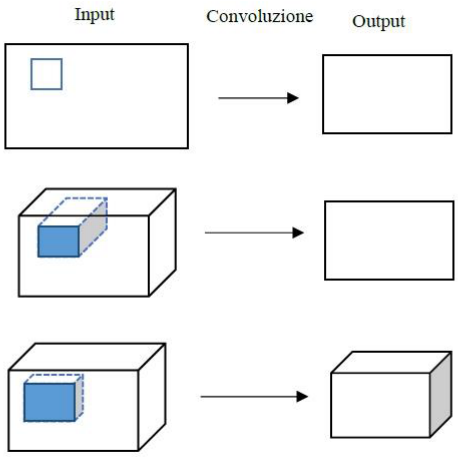

Figura 3.4.: Esempi di convoluzione 2D e 3D tratto da [1]
Partendo dall'alto troviamo una convoluzione 2D e un input rappresentato da una singola immagine. Al centro si hanno più frame in input per una convoluzione 2D. Infine, in basso viene rappresentata una convoluzione 3D.

Le reti convoluzionali 3D [4] (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 [4]. 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 [12]. 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*.

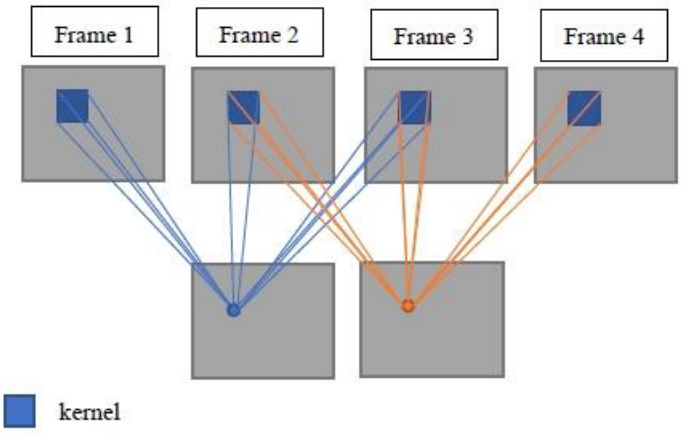

Figura 3.5.: Esempio di convoluzione 3D con 4 frame in input approssimato in 2D tratto da [1]

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.

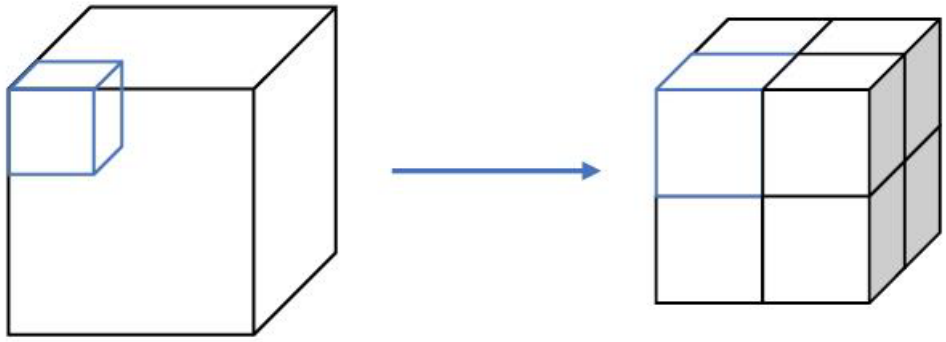

Figura 3.6.: Esempio di 3D Pooling tratto da [1]

3.2.2. Modello di rete C3D

Come anticipato, i due modelli utilizzati in questo lavoro di tesi, si basano su una rete convoluzionale 3D. C3D [4] 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.

3.2.3. 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'architetttura con

Support Vector Machine per la classificazione, si è optato di non utilizzare questi layer.
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.

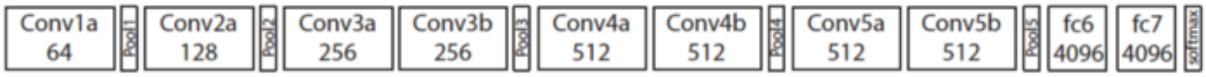

Figura 3.7.: Architettura C3D tratta da [4]

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.

3.2.4. 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. 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 andremo ad illustrare dettagliatamente il sistema realizzato. Inizieremo facendo una panoramica della piattaforma cloud utilizzata per implementare i nostri modelli e delle varie librerie che sono state necessarie al nostro scopo. In seguito, proseguiremo mostrando nel dettaglio i sette 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.*". *Keras* è stato utilizzato da Cappanera [6] 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, si è deciso di riutilizzare questi tre famosi dataset e in aggiunta altri 3 dataset: AIRTLab, Surveillance Camera Fight Dataset [8] e il Real Life Violence Situation Dataset [7].
Come era stato sottolineato nel capitolo precedente, l'architettura proposta lavora con degli input da 16 frame ciascuno da cui estrae le feature. 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 sei 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 non è bilanciato, infatti abbiamo 120 video raffiguranti comportamenti normali e 230 video raffiguranti scene di violenza. A loro volta le clip di ogni classe vengono divise in due sottocartelle (cam1 e cam2) che contengono lo stesse numero di 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.

4.2.5. Surveillance Camera Fight Dataset

Il Surveillance Camera Fight Dataset [8] è stato realizzato collezionando video da Youtube contenenti scene di violenza. Inoltre sono stati aggiunti anche video con sequenza di frame non violenti registrati dai regolari sistemi di videosorveglianza. Il dataset è composto da 300 video, ben bilanciato con 150 video con scene di violenza e 150 video

con scene di comportamenti normali. Tutti i filmati sono memorizzati con codifica mp4 e la lunghezza media di ogni video è di 2 secondi. Inoltre, sono inclusi vari scenari di combattimento come colpire con un oggetto, calci, pugni, wrestling. Anche l'ambiente nei campioni varia come caffè, strada, autobus e altro.
Il dataset è suddiviso in due singole classi (fight e no-fight) ed ogni clip è stata etichettata con la classe di appertenenza. L'etichetta, comunque, è caratterizzata da una stringa che rispetta la seguente espressione regolare:

$$(fi/nofi) + ([0-9]\{1,3\}) + .mp4$$

Dove i primi due caratteri rappresentano la classe etichettata: "fi" in caso di aggressione e "nofi" in caso di nessuna aggressione. I numeri, invece, indicano l'indice del video rispetto ad una classe.

4.2.6. Real Life Violence Situation Dataset

Il Real Life Violence Situation Dataset [7] contiene 2000 video presi da Youtube, contenenti vere situazioni di combattimento di strada in diversi ambienti e condizioni, anche i video non violenti sono raccolti da molte diverse azioni umane come sport, mangiare, camminare.
Questo dataset risulta fortemente etichettato e bilanciato, infatti abbiamo 1000 video con scene di violenza e 1000 video con comportamenti normali. Tutti i filmati hanno una durata di 5 secondi con 30 FPS e una codifica mp4, in questo caso però non abbiamo una risoluzione standard. Anche in questo caso, il dataset è suddiviso in due singole classi (Violence e NonViolence) e l'etichetta di ogni clip è caraterizzata da una stringa che rispetta la seguente espressione:

$$(V/NV)\_([0-9]\{1,3\}) + .mp4$$

Dove i primi due caratteri rappresentano la classe etichettata: "V" in caso di aggressione e "NV" in caso di nessuna aggressione. I numeri, invece, indicano l'indice del video rispetto ad una classe.

4.3. Sistema di Riconoscimento modello C3D + SVM

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

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

Analizziamo nel dettaglio queste tre componenti che formano il sistema di riconoscimento.

4.3.1. Preprocessing dell'Input

La rete C3D, come già detto precedentemente, processa input di 16 frame e per questo si è deciso di suddividere ogni video in segmenti di tale dimensioni. Come detto nei paragrafi precedenti, i video dei vari dataset di benchmark hanno una lunghezza diversa, perciò va definito come sfruttare l'input. Inoltre, la violenza all'interno dei video non è continua, ma potrebbero esserci delle pause tra una scena e un'altra. Questo implica che potrebbero esserci sequenze di frame all'interno dei filmati etichettati come violenti in cui effettivamente la violenza non è presente.

In letteratura, molti degli approcci che vengono usati producono un campionamento dei frame dei video, partendo da un istante casuale e campionando sequenzialmente. In questo lavoro, abbiamo deciso 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 Descriptor

L'elemento centrale del sistema di riconoscimento è l'estrattore di feature. Come descritto 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 illustrato nei capitoli

precedenti, il classificatore che è stato scelto per tale scopo è una SVM. 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.

A differenza del lavoro di Accattoli [1], dove si utilizzava solo una SVM lineare, in questo progetto di tesi si è voluto implementare il Grind Search, che è una tecnica di ottimizzazione degli iperparametri del modello. In Keras questa tecnica viene fornita nella classe GridSearchCV.
Uno dei motivi per cui si è optato per il GrindSearchCV è perchè esso è un meta-estimatore, cioè prende uno stimatore, nel nostro caso SVC, e crea un nuovo stimatore, che si comporta esattamente allo stesso modo, in questo caso, come classificatore.
Il GridSearchCV prende un dizionario che descrive i parametri che potrebbero essere provati su un modello per addestrarlo. La griglia dei parametri è definita come un dizionario, dove le chiavi sono i parametri e i valori sono le impostazioni da testare. I parametri variati durante il test, come si vede dall'immagine 4.1, sono:

- kernel 'linear' e 'rbf';
- la variabile 'C' con i valori '1, 10, 100, 1000'.

```
param_grid =[{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
              'C': [1, 10, 100, 1000]},
             {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
grid = GridSearchCV(SVC(probability=True), param_grid, n_jobs=-1, cv=cv)
```

Figura 4.1.: Params grid

Dopo aver costruito il dataset, si passa all'addestramento del classificatore andando a dividere 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 dell'intero sistema di riconoscimento può essere rappresentata come segue:

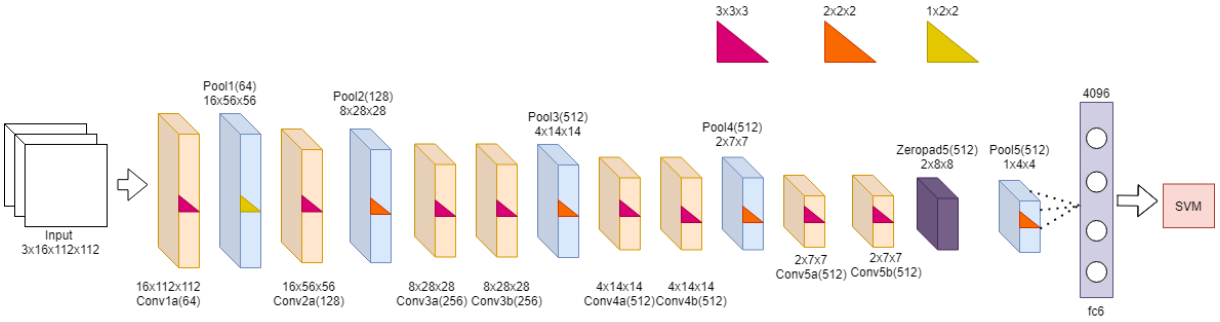

Figura 4.2.: Architettura del sistema di riconoscimento

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 ripreso il porting effettuato da Cappanera [6] nel suo lavoro di tesi. Questo porting è basato su una responsity 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 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 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 4.3.: 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 4.4.: 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 4.5.: Codice relativo all'estrazione e salvataggio su file delle feature

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 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. 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 4.6.: Implementazione della funzione write_dataset

Infine, andiamo a vedere come viene eseguita la classificazione. 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*.

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

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


tprs = []
aucs = []
scores = []
mean_fpr = np.linspace(0, 1, 100)
plt.figure(num=1, figsize=(10,10))
i = 1
for train, test in cv.split(X, y):
    param_grid =[{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                  'C': [1, 10, 100, 1000]},
                 {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
    grid = GridSearchCV(SVC(probability=True), param_grid, n_jobs=-1, cv=cv)
    grid_result = grid.fit(X[train], y[train])
    pred_acc = grid.predict(X[test])

    scores.append(accuracy_score(y[test], pred_acc))
    prediction = grid.predict_proba(X[test])
```

Figura 4.7.: Classificatore

4.4. Sistema di Riconoscimento modello C3D end-to-end

Per realizzare il sistema di riconoscimento sono stati due gli elementi principali su cui ci siamo concentrati:

- *Preprocessing dell'input.*
- *Scelta degli optimizer.*

Andremo ad analizzare nel dettaglio solo la scelta degli optimizer.
Per quanto riguarda il preprocessing, essendo anche questo modello basato su una rete C3D, si è deciso di suddividere ogni video in una sequenza di 16 frame, e contare di quante sequenze è formato ogni video. I campioni e le etichette vengono archiviati su due array numpy memmap, "*samples.mmap*" and "*labels.mmap*", questo per evitare di caricare tutti i campioni in memoria contemporaneamente.

4.4.1. Scelta degli optimizer

Un optimizer è un *Optimization algorithm*, ovvero un algoritmo di ottimizzazione che permette d'individuare, attraverso una serie d'iterazioni, quei valori dei weight tali per cui la cost function risulti avere il valore minimo.
I tre optimizer scelti in questo lavoro di tesi sono:

– Adam

– AdaDelta

– RMSProp

Adam

L'algoritmo di ottimizzazione *Adam* è un'estensione della discesa del gradiente stocastico che ha recentemente visto un'adozione più ampia per le applicazioni di deep learning nella visione artificiale e nell'elaborazione del linguaggio naturale.
I vantaggi dell'utilizzo di *Adam*, così come riportano gli autori [37], su problemi di ottimizzazione non convessi, sono:

- Semplice da implementare.
- Computazionalmente efficiente.
- Pochi requisiti di memoria.
- Invariante al ridimensionamento diagonale dei gradienti.
- Adatto per problemi di grandi dimensioni in termini di dati e/o parametri.
- Adatto per obiettivi non stazionari.
- Appropriato per problemi con gradienti molto rumorosi/o sparsi.
- Gli iperparametri hanno un'interpretazione intuitiva e in genere richiedono poca regolazione.

Nello specifico, l'algoritmo calcola una media mobile esponenziale del gradiente e del gradiente al quadrato, e i parametri beta1 e beta2 controllano i tassi di decadimento di queste medie mobili. Il valore iniziale delle medie mobili e i valori di beta1 e beta2 vicini a 1,0 (consigliato) determinano una distorsione delle stime del momento verso lo zero. Questa distorsione viene superata calcolando le stime distorte prima di calcolare le stime corrette per la distorsione.

AdaDelta

Adadelta [38] è un'estensione di Adagrad che cerca di ridurre il suo tasso di apprendimento aggressivo e monotono decrescente basato su una finestra mobile fissa di aggiornamenti del gradiente, invece di accumulare tutti i gradienti passati.
Adadelta funziona come un metodo stocastico di discesa del gradiente.

RMSProp

Root Mean Square Propagation(RMSProp) è un'estensione inedita [39] dell'algoritmo di ottimizzazione della discesa del gradiente. *RMSProp* è progettato per accelerare il processo di ottimizzazione, ad es. diminuire il numero di valutazioni della funzione necessarie per raggiungere l'optima o per migliorare la capacità dell'algoritmo di ottimizzazione.

4.4.2. Implementazione in Google Colab

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

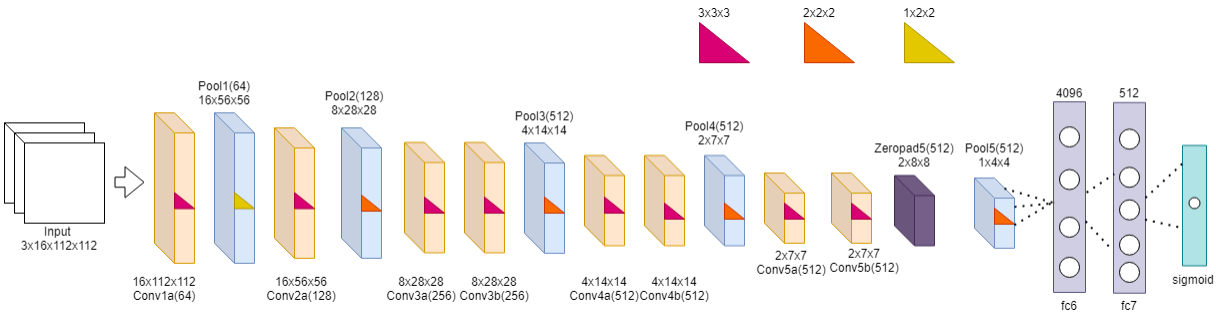

Figura 4.8.: Architettura del sistema di riconoscimento end-to-end

Come si evince dall'immagine il modello è una combinazione di C3D come estrattore di funzionalità e due livelli completamente connessi per ottenere una rete end-to-end per la classificazione. Anche in questo modello sono stati utilizzati i pesi originali C3D senza allenamento di nuovo, applicando il transfer learning. Solo i due livelli finali completamente connessi vengono addestrati da zero sulle clip dei set di dati.

```python
def getC3DCNNModel(optimizer):

    pretrainedModel = initialize_feature_extractor()
    for layer in pretrainedModel.layers:
        layer.trainable = False

    dropout1 = Dropout(.5)(pretrainedModel.output)
    fc7Alt = Dense(512, activation='relu', name='fc7-alt')(dropout1)
    dropout2 = Dropout(.5)(fc7Alt)
    output = Dense(1, activation='sigmoid')(dropout2)
    model = Model(inputs=pretrainedModel.inputs, outputs=output)
    model.summary()
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

    del pretrainedModel

    return model
```

Figura 4.9.: Codice relativo all'implementazione della rete C3D

Le fasi di preprocessing dell'input e la creazione delle numpy memmap vengono realizzate in un'unica funzione chiamata *preprocessVideos*, che prede come parametri il path del percorso alla base del repository video ed il percorso in cui verranno salvati gli array numpy. Prima di dividere i video nelle sequenze da 16 frame, con la funzione *count_chunks* e l'ausilio delle funzioni della libreria *OpenCV*, si calcola il numero di blocchi di 16 frame disponibili nel dataset, così da creare la memmap con le dimensini appropriate per il dataset selezionato. A questo punto si procede con il creare i segmenti di frame, e memorizzarli nelle memmap etichettandoli a seconda della classe di appartenza, l'aggressione viene identificata con la label 1, mentre l'assenza di aggressione con la label 0.

```python
def preprocessVideos(directory, featureBasePath, verbose=True):

    total_chunks = count_chunks(directory)
    npSamples = np.memmap(os.path.join(featureBasePath, 'samples.mmap'), dtype=np.float32, mode='w+', shape=(total_chunks, 16, 112, 112, 3))
    npLabels = np.memmap(os.path.join(featureBasePath, 'labels.mmap'), dtype=np.int8, mode='w+', shape=(total_chunks))
    cnt = 0

    for subdir in listdir(directory):
        path = os.path.join(directory, subdir)
        videofiles = os.listdir(path)
        for videofile in videofiles:
            filePath = os.path.join(path, videofile)
            video = cv2.VideoCapture(filePath)
            numframes = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
            fps = int(video.get(cv2.CAP_PROP_FPS))
            chunks = numframes//16
            if verbose:
              print(filePath)
              print("*** [Video Info] Number of frames: {} - fps: {} - chunks: {}".format(numframes, fps, chunks))
            vid = []
            videoFrames = []
            while True:
                ret, img = video.read()
                if not ret:
                    break
                videoFrames.append(cv2.resize(img, (112, 112)))
            vid = np.array(videoFrames, dtype=np.float32)
            filename = os.path.splitext(videofile)[0]
            chunk_cnt = 0
            for i in range(chunks):
                X = vid[i*16:i*16+16]
                chunk_cnt += 1
                npSamples[cnt] = np.array(X, dtype=np.float32)
                if (subdir == 'aggressione' or subdir == 'fights' or subdir == 'fight' or subdir == 'cam1_violent'
                    or subdir == 'cam2_violent' or subdir == 'violence' or subdir == 'Violence'):
                    npLabels[cnt] = np.int8(1)
                else:
                    npLabels[cnt] = np.int8(0)
                cnt += 1

    if verbose:
        print("** Labels **")
        print(npLabels.shape)
        print('\n****\n')
        print("** Samples **")
        print(npSamples.shape)
        print('\n****\n')

    del npSamples
    del npLabels
```

Figura 4.10.: Codice relativo alla creazione delle numpy memmap

Di seguito andiamo a definire il codice per eseguire gli esperimenti sul modello. Come con C3D + SVM, gli esperimenti sono test ripetuti 5 volte con lo schema di convalida incrociata Stratified Shuffle Split. In ogni suddivisione l'80% dei dati viene utilizzato per l'addestramento e il 20% dei dati viene utilizzato per i test. 12,5% dei dati di addestramento (ovvero il 10% dell'intero set di dati) viene utilizzato per la convalida. In altre parole, in ogni test il 70% dei dati è effettivamente per la formazione, il 10% per la convalida e il 20% per il test.

```
def runEndToEndExperiment(getModel, batchSize, datasetBasePath, mmapDatasetBasePath, samplesMMapName, lablesMMapName, endToEndModelName):

    chunk_number = count_chunks(datasetBasePath)
    optimizer = ['Adam', 'Adadelta', 'RMSprop']
    epochs=[20, 50]
    for opt in optimizer:
        print('Modello con optmizier: ' + opt)
        for ep in epochs:
            print('Test con numero di epochs: ', ep, '\n' )


            X = np.memmap(os.path.join(mmapDatasetBasePath, samplesMMapName), mode='r', dtype=np.float32, shape=(chunk_number, 16, 112, 112, 3))
            y = np.memmap(os.path.join(mmapDatasetBasePath, lablesMMapName), mode='r', dtype=np.int8, shape=(chunk_number))

            nsplits = 5

            cv = StratifiedShuffleSplit(n_splits=nsplits, train_size=0.8)

            tprs = []
            aucs = []
            scores = []
            sens = np.zeros(shape=(nsplits))
            specs = np.zeros(shape=(nsplits))
            f1Scores = np.zeros(shape=(nsplits))
            mean_fpr = np.linspace(0, 1, 100)
            plt.figure(num=1, figsize=(10,10))
            i = 1

            for train, test in cv.split(X, y):

                X_train = np.memmap(os.path.join(mmapDatasetBasePath, 'samples_train.mmap'), mode='w+', dtype=np.float32, shape=X[train].shape)
                X_train[:] = X[train][:]

                X_test = np.memmap(os.path.join(mmapDatasetBasePath, 'samples_test.mmap'), mode='w+', dtype=np.float32, shape=X[test].shape)
                X_test[:] = X[test][:]

                del X

                model = getModel(opt)

                es = EarlyStopping(monitor='val_loss', mode='min', patience=5, verbose=1, restore_best_weights=True)
                model.fit(X_train, y[train], validation_split=0.125, epochs=ep, batch_size=batchSize, verbose=1, callbacks=[es])
```

Figura 4.11.: Codice relativo alla funzione per gli eseguire gli esperimenti

Si è deciso di implementare anche una funzione callback di EarlyStopping, un metodo che consente di specificare un numero arbitrariamente elevato di epoche di addestramento e di interromperlo una volta che le prestazioni del modello smettono di migliorare nel set di dati di convalida. Spesso, il primo segno di nessun miglioramento potrebbe non essere il momento migliore per interrompere l'allenamento. Questo perché il modello potrebbe peggiorare leggermente prima di migliorare notevolmente. Possiamo spiegarlo aggiungendo un ritardo al trigger in termini di numero di epoche su cui non vorremmo vedere alcun miglioramento. Il valore monitorato è stato il "*val_loss*" e il numero di epochs di ritardo è

stato impostato a 5.

I parametri che la funzione per eseguire gli esperimenti prende in ingresso sono:

- il modello;
- dimensione del Batch size da utilizzare per il training ed il test;
- il path del percorso alla base del repository video;
- cartella contenente le memory map;
- nome del file che contiene l'array numpy dei campioni;
- nome del file che contiene l'array numpy delle etichette;
- nome del modello da utilizzare nel grafico AUC-ROC.

Capitolo 5.

Esperimenti e Risultati

Il seguente capitolo illustra i vari test effettuati sui sette dataset di benchmark. Andando a valutare il metodo proposto, con relativa analisi riguardando i risultati ottenuti. Infine, verrà mostrato un'analisi riguardanti gli errori ottenuti dalla rete.

5.1. Setting Sperimentale C3D + SVM

Per valutare le performance, si è deciso di utilizzare come metriche di valutazione l'accuratezza e l'area under the curve (AUC), sfruttando lo schema della 5-fold cross validation. La scelta è ricaduta per la 5-fold per rimanere conforme agli approcci appartenenti allo stato dell'arte. Ogni dataset è diviso in cinque differenti split, dove quattro dei quali vengono utilizzati per il training e il restante per il testing. Questo viene ripetuto cambiando il test set per ognuno dei 5 split.

Il caricamento in memoria del dataset risultava essere ordinato per classe, perciò è stato necessario utilizzare lo *shuffling*. Senza il rimescolamento gli split prodotti risultavano essere molto sbilanciati e questi non permettevano un addestramento corretto, riducendo notevolmente l'accuratezza.

Tramite la K-fold l'accuracy viene calcolata ad ogni iterazione e il risultato finale è rappresentato dalla media con la relativa deviazione standard. In aggiunta all'accuratezza, si è adottata come ulteriore misura, 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 (errori di tipo 1).

Il confronto nel Movie Violence Dataset non viene riportato in quanto caratterizzato da un numero molto limitato di campioni, ma soprattutto essi risultano molto facili da discriminare. A dimostrazione di questo il metodo proposto ottiene il 100% di accuratezza con una deviazione standard di 0. Questo risultato potrebbe sembrare errato, ma anche altri approcci raggiungono un'accuratezza di questo tipo [16]. In aggiunta a questo il dataset è poco utilizzato.

5.1.1. Test Hockey Fight Dataset

Nell'immagine sottostante riportiamo i risultati ottenuti nei nostri test tramite il Grind Search.

```
Best: 0.970616 using {'C': 1, 'kernel': 'linear'}
Mean test score (std test score) with : params
0.761137 (0.024871) with: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.958768 (0.006106) with: {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.776303 (0.025015) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.970616 (0.005104) with: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.776303 (0.025015) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.970616 (0.005104) with: {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.776303 (0.025015) with: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.970616 (0.005104) with: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.970616 (0.008290) with: {'C': 1, 'kernel': 'linear'}
0.970616 (0.008290) with: {'C': 10, 'kernel': 'linear'}
0.970616 (0.008290) with: {'C': 100, 'kernel': 'linear'}
0.970616 (0.008290) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.1.: Risultati test Hockey Fight

Come si può osservare dall'immagine, il risultato migliore si è ottenuto con un kernel lineare. I valori risultano uguali perchè le Linear Support Machine Vector sono meno sensibili a "C" quando diventa grande ed i risultati della previsione smettono di migliorare dopo una certa soglia.
Di seguito riportiamo anche la matrice di confusione relativa ad un singolo test con split casuale. Come illustrato ad inizio capitolo, anche qui lo split prevedeva l'utilizzo del 70% dei campioni per il training e del restante 30% per il testing.

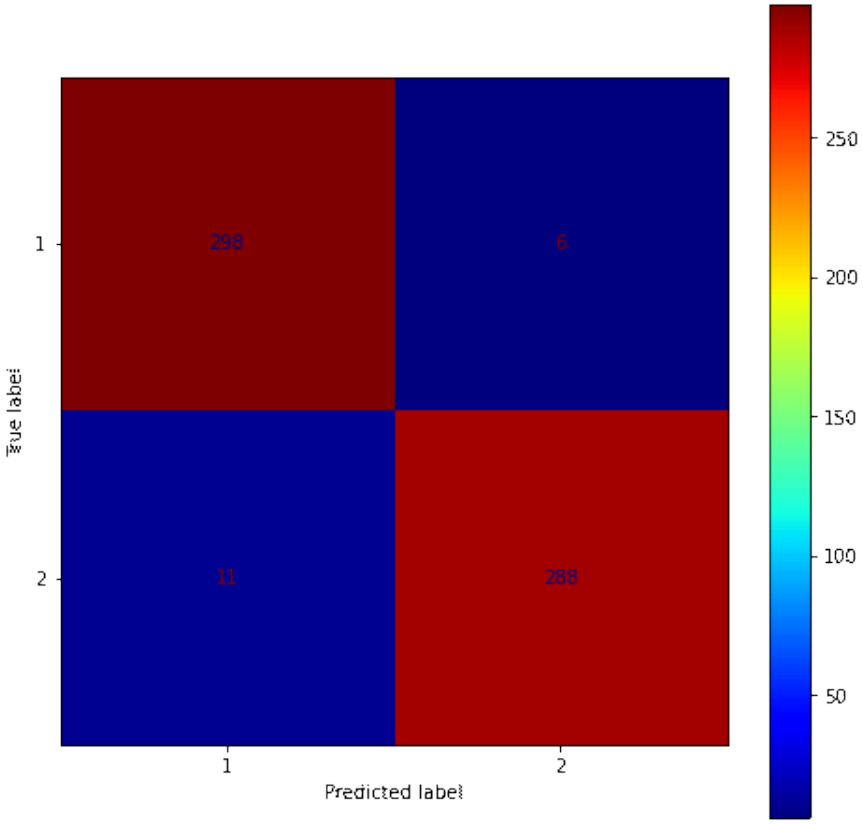

Figura 5.2.: Matrice di confusione ottenuta da un singolo test sull'Hockey Fight Dataset

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 97,74%.

Nella prossima immagine, invece, è raffigurata la curva AUC prodotta tramite 5-fold cross validation.

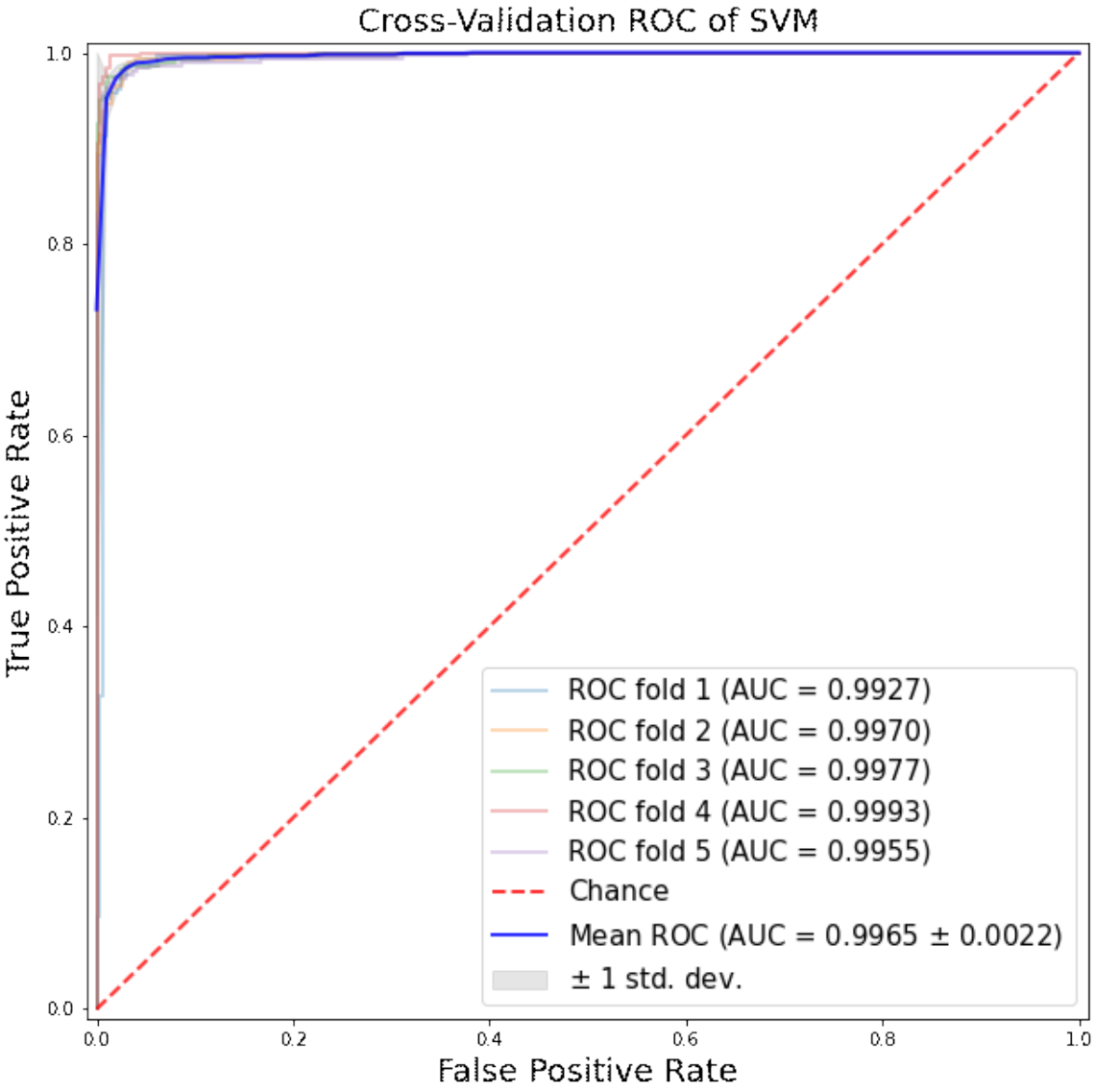

Figura 5.3.: Area Under Curve dell'Hockey Fight Dataset

5.1.2. Test Crowd Violence Dataset

Il Crowd dataset risulta essere una base di dati molto competitiva per via delle seguenti caratteristiche:

- Scarsa risoluzione dei filmati
- Pochi campioni

- Presenza di molte persone che tendono a sovrapporsi risultando difficile l'estrazione di feature discriminanti.

```
Best: 0.990189 using {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
Mean test score (std test score) with : params
0.702642 (0.044419) with: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.980377 (0.005546) with: {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.726038 (0.037402) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.990189 (0.006999) with: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.726038 (0.037402) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.990189 (0.006999) with: {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.726038 (0.037402) with: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.990189 (0.006999) with: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.989434 (0.005546) with: {'C': 1, 'kernel': 'linear'}
0.989434 (0.005546) with: {'C': 10, 'kernel': 'linear'}
0.989434 (0.005546) with: {'C': 100, 'kernel': 'linear'}
0.989434 (0.005546) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.4.: Risultati test Crowd Violence Dataset

Come si evince dall'immagine il risultato migliore lo si è ottenuto con un kernel di tipo "rbf" e un valore di gamma pari a 10^{-4}. Il sistema inoltre ha il grande vantaggio di produrre pochi falsi negativi, che ricordiamo essere il caso peggiore. Ad esempio, nel test relativo alla matrice di confusione sottostante i casi negativi vengono predetti correttamente.

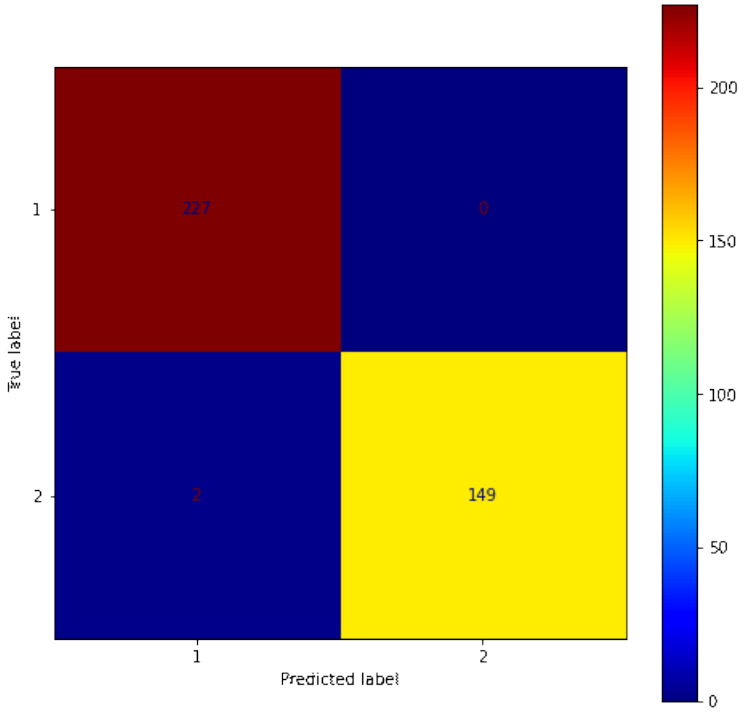

Figura 5.5.: Matrice di confusione ottenuta da un singolo test sul Crowd Violence Dataset

Il dataset risulta essere di dimensione minore per via del numero limitato di video (246). Al contrario dell'hockey fight, il crowd violence non è perfettamente bilanciato per via

della lunghezza variabile dei filmati. La curva AUC ottenuta dalla 5-fold è rappresentata dalla seguente figura.

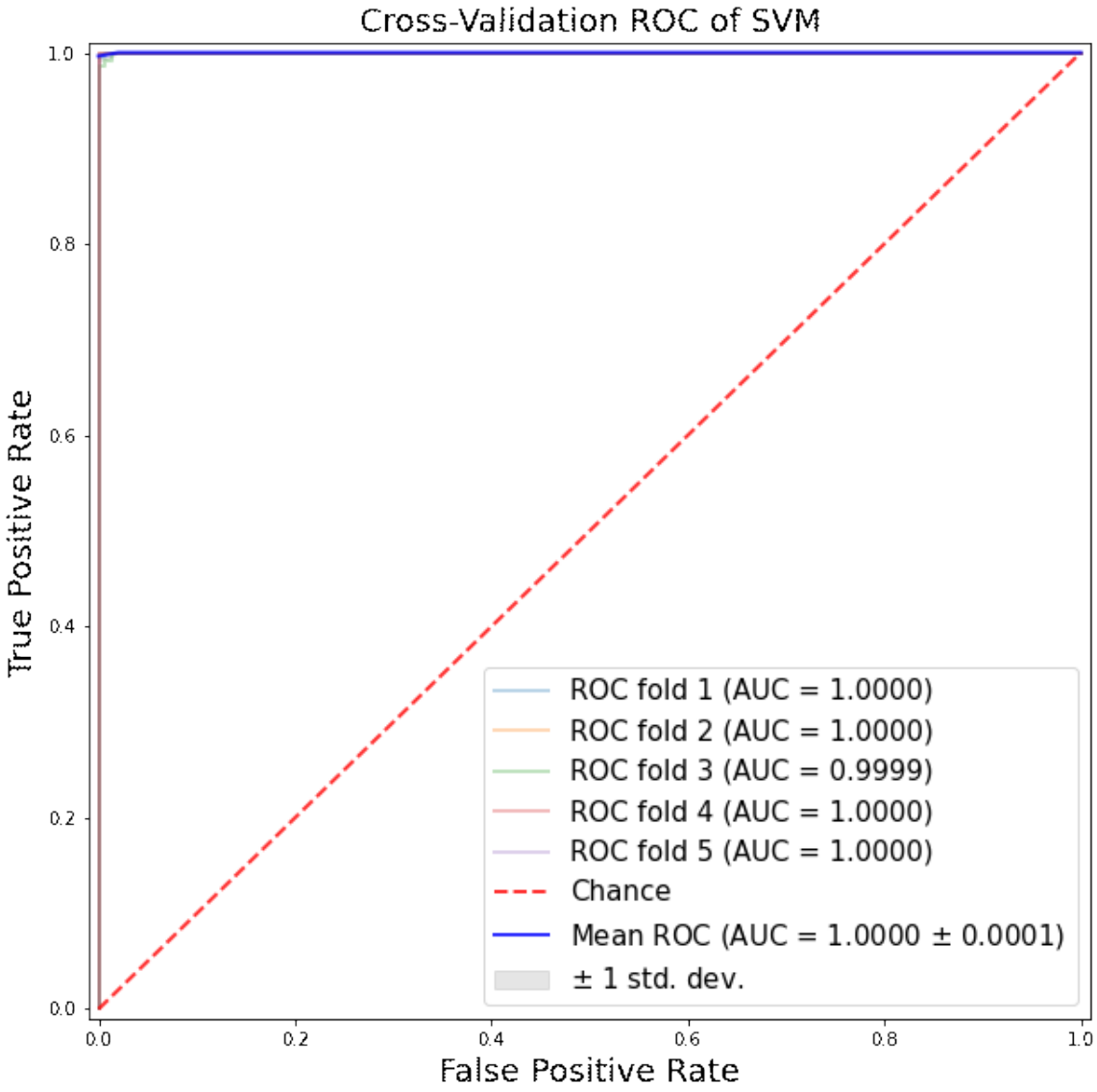

Figura 5.6.: Area Under Curve del Crowd Violence Dataset

5.1.3. Test Dataset Intero

Per valutare la capacità di generalizzazione del sistema realizzato, esso è stato testato con un'ulteriore base di dati rappresentata dalla collezione dei tre dataset di benchmark famosi in letteratura, 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. Il problema di utilizzarlo in queste condizioni è il riuscire a rappresentare in maniera esaustiva il caso di comportamento normale.

```
Best: 0.976309 using {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
Mean test score (std test score) with : params
0.746883 (0.041196) with: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.963840 (0.005349) with: {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.771072 (0.039134) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.976309 (0.005171) with: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.771072 (0.039134) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.976309 (0.005171) with: {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.771072 (0.039134) with: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.976309 (0.005171) with: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.972818 (0.004276) with: {'C': 1, 'kernel': 'linear'}
0.972818 (0.004276) with: {'C': 10, 'kernel': 'linear'}
0.972818 (0.004276) with: {'C': 100, 'kernel': 'linear'}
0.972818 (0.004276) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.7.: Risultati test sul Dataset Intero

L'accuratezza ottenuta è di 97,6%, con il test eseguito con il kernel "rbf" e gamma pari a 10^{-4}. Di seguito riportiamo la matrice di confusione, rispetto ad un campionamento casuale.

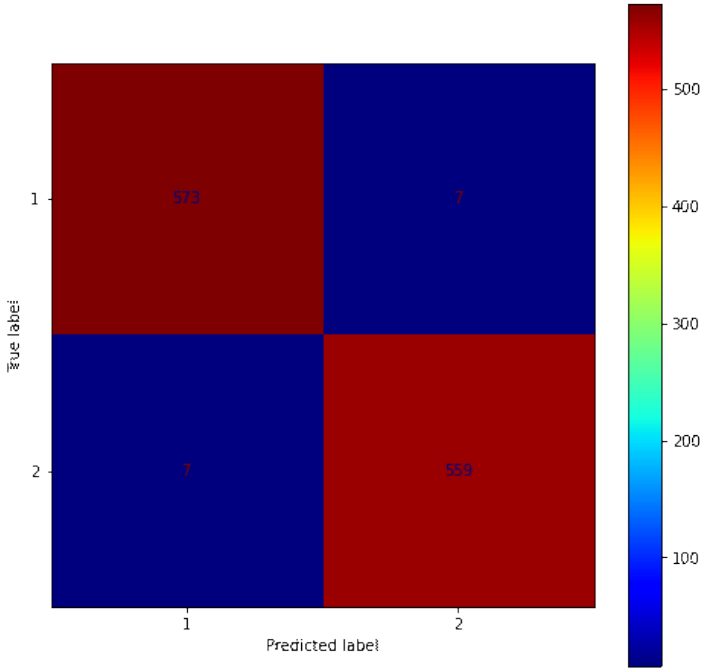

Figura 5.8.: Matrice di confusione ottenuta da un singolo test sul Dataset Intero

La curva AUC ottenuta dalla 5-fold è rappresentata dalla seguente figura.

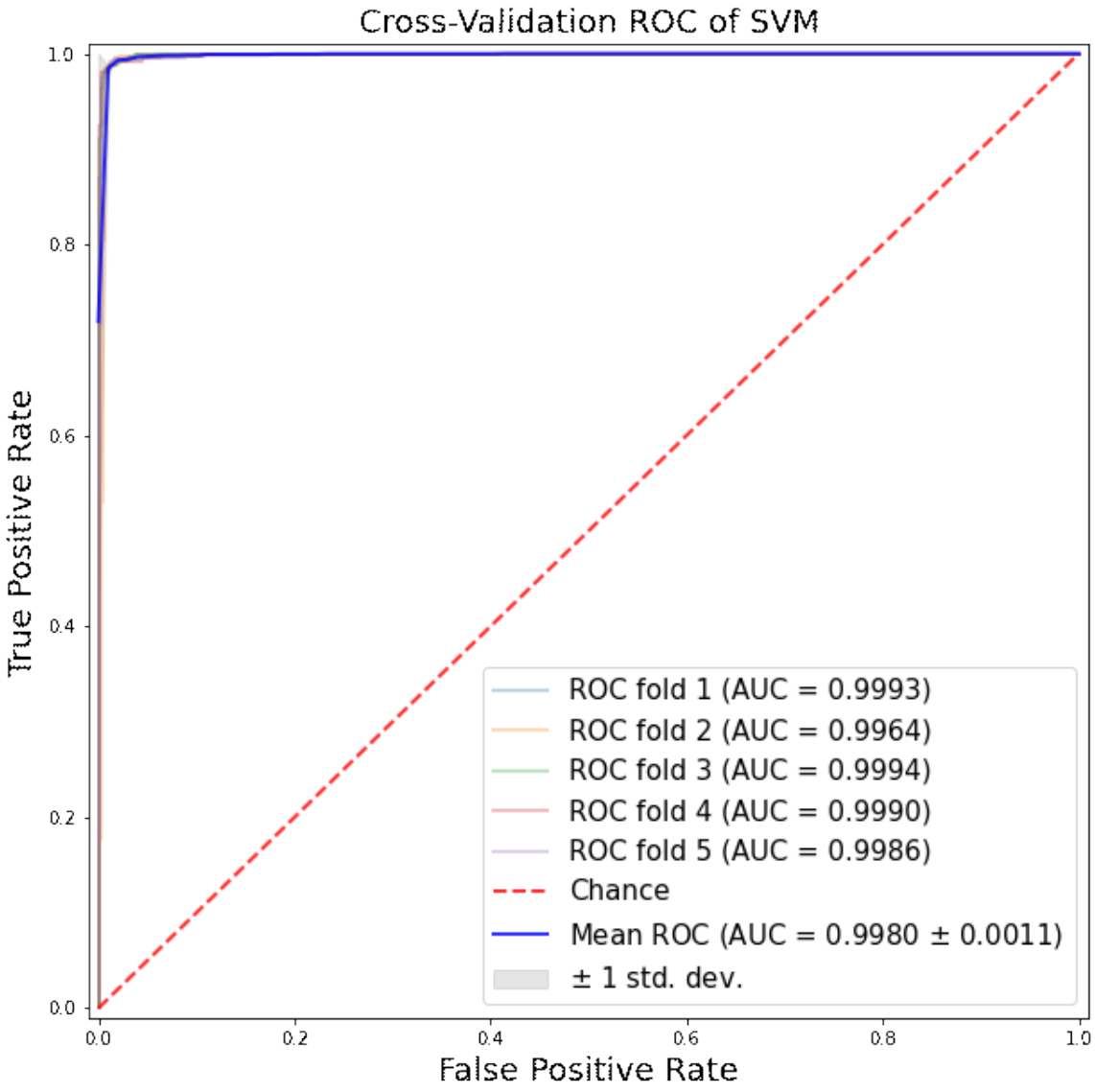

Figura 5.9.: Area Under Curve del Dataset Intero

5.1.4. 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 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 comporta il raggiungimento di un'accuratezza media minore rispetto ai test precedenti.

```
Best: 0.960700 using {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
Mean test score (std test score) with : params
0.840646 (0.012982) with: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.918170 (0.005065) with: {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.862988 (0.008048) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.959892 (0.004290) with: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.862988 (0.008048) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.960700 (0.007243) with: {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.862988 (0.008048) with: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.960700 (0.007243) with: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.940242 (0.007921) with: {'C': 1, 'kernel': 'linear'}
0.940242 (0.007921) with: {'C': 10, 'kernel': 'linear'}
0.940242 (0.007921) with: {'C': 100, 'kernel': 'linear'}
0.940242 (0.007921) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.10.: Risultati test sull' AIRTLab Dataset

Come si può notare dall'immagine l'accuratezza è pari al 96%, con un kernel di tipo "rbf" e un valore di gamma pari a 10^{-4}.
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.

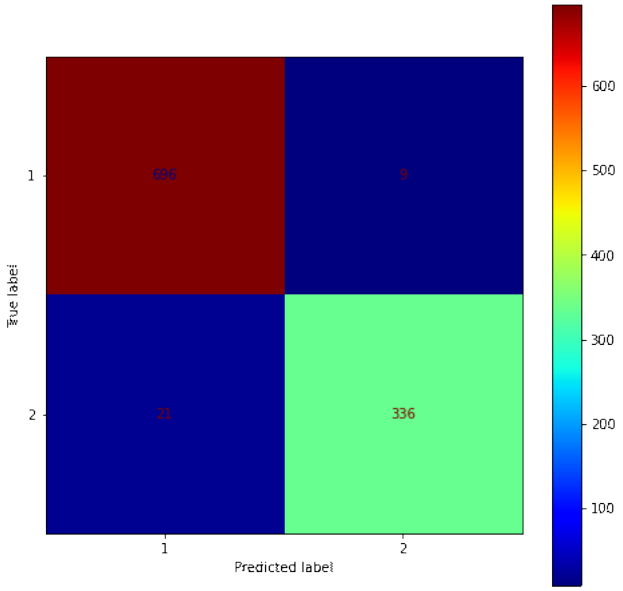

Figura 5.11.: Matrice di confusione ottenuta da un singolo test sul AIRTLab Dataset

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

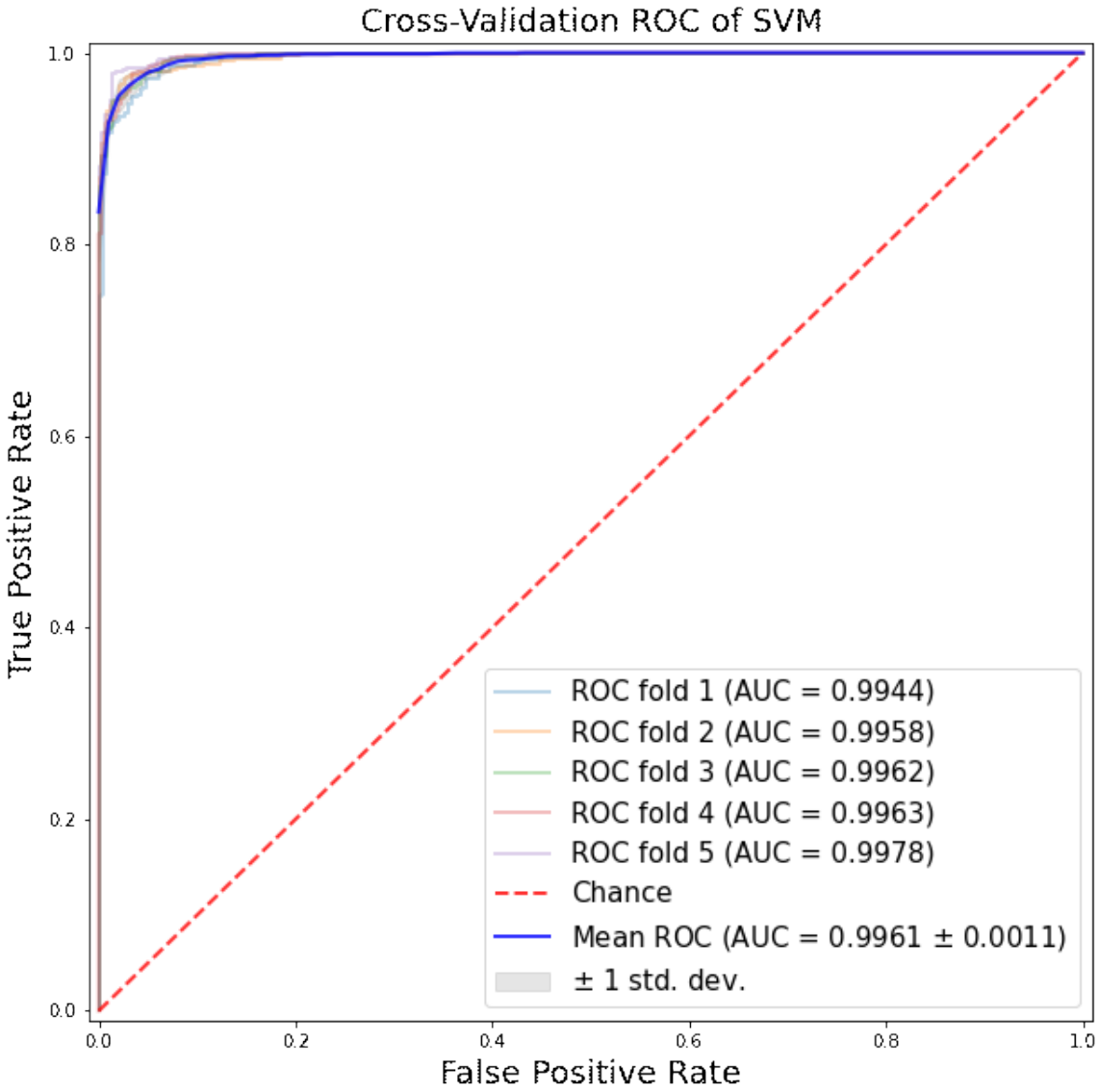

Figura 5.12.: Area Under Curve del AIRTLab Dataset

5.1.5. Test Surveillance Camera Fight Dataset

Nell'immagine sottostante riportiamo i risultati ottenuti dal test effetuato.

```
Best: 0.916667 using {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
Mean test score (std test score) with : params
0.820833 (0.045238) with: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.867708 (0.013421) with: {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.834375 (0.043626) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.916667 (0.021850) with: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.834375 (0.043626) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.916667 (0.021850) with: {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.834375 (0.043626) with: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.916667 (0.021850) with: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.900000 (0.020937) with: {'C': 1, 'kernel': 'linear'}
0.900000 (0.020937) with: {'C': 10, 'kernel': 'linear'}
0.900000 (0.020937) with: {'C': 100, 'kernel': 'linear'}
0.900000 (0.020937) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.13.: Risultati test sul Surveillance Camera Fight Dataset

Come si può osservare, l'accuratezza migliore si è ottenuta con un kernel "rbf" e un valore di gamma pari a 10^{-4}.
Il dataset, come detto nel capitolo precedente, è fortemente bilanciato e come possiamo vedere dalla matrice di confusione, riportata di seguito, i falsi positivi e falsi negativi hanno valori molto bassi.

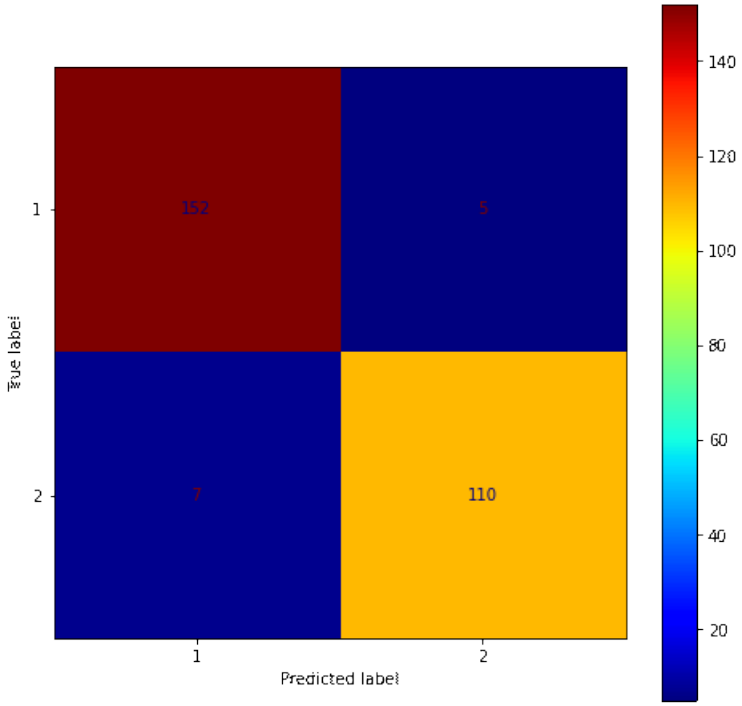

Figura 5.14.: Matrice di confusione ottenuta da un singolo test sul Surveillance Camera Fight Dataset

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

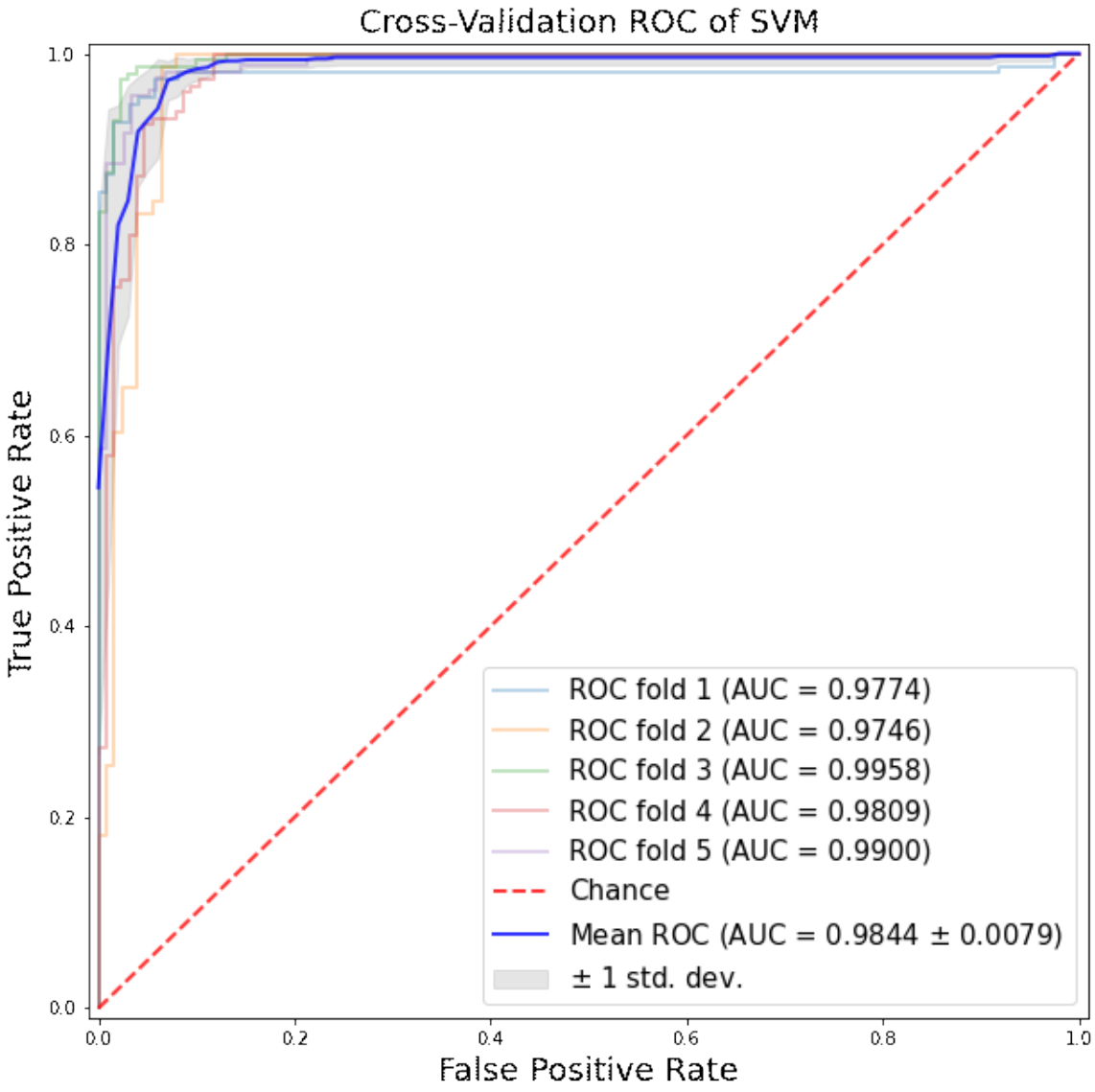

Figura 5.15.: Area Under Curve del Surveillance Camera Fight Dataset

5.1.6. Test Real Life Violence Dataset

Il Real Life Violence Dataset è il dataset con più filmati testato in questo progetto, 2000 video in totale.

```
Best: 0.992926 using {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
Mean test score (std test score) with : params
0.992121 (0.002341) with: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.992926 (0.002855) with: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.982511 (0.002702) with: {'C': 1, 'kernel': 'linear'}
0.982511 (0.002702) with: {'C': 10, 'kernel': 'linear'}
0.982511 (0.002702) with: {'C': 100, 'kernel': 'linear'}
0.982511 (0.002702) with: {'C': 1000, 'kernel': 'linear'}
```

Figura 5.16.: Risultati test sul Real Life Violence Dataset

Purtroppo, come si vede dall'immagine sopra riportata, per i limiti imposti da Google Colab, non abbiamo potuto effettuare tutti i test, come nei precedenti dataset. Il risultato migliore si è ottenuto con un kernel di tipo "rbf" e un valore di gamma pari a 10^{-4}. Come si evince dalla matrice di confusione sottostante, il sistema ha un'accuratezza molto elevata, con pochi falsi positivi e negativi.

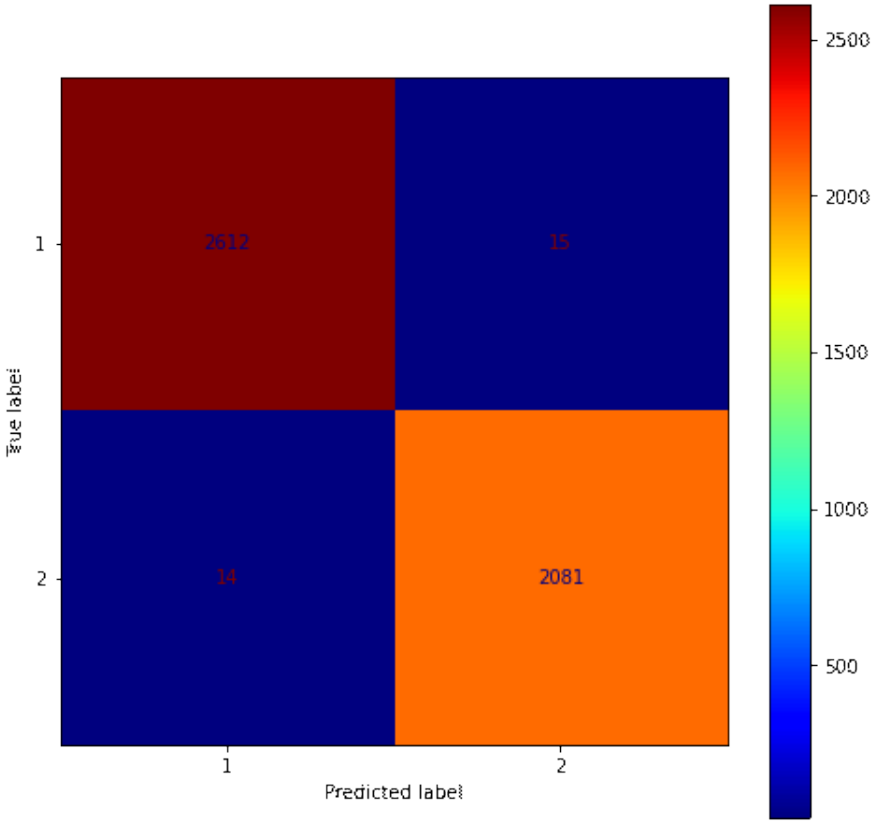

Figura 5.17.: Matrice di confusione ottenuta da un singolo test sul Real Life Violence Dataset

La curva AUC ottenuta dalla 5-fold è rappresentata dalla seguente figura.

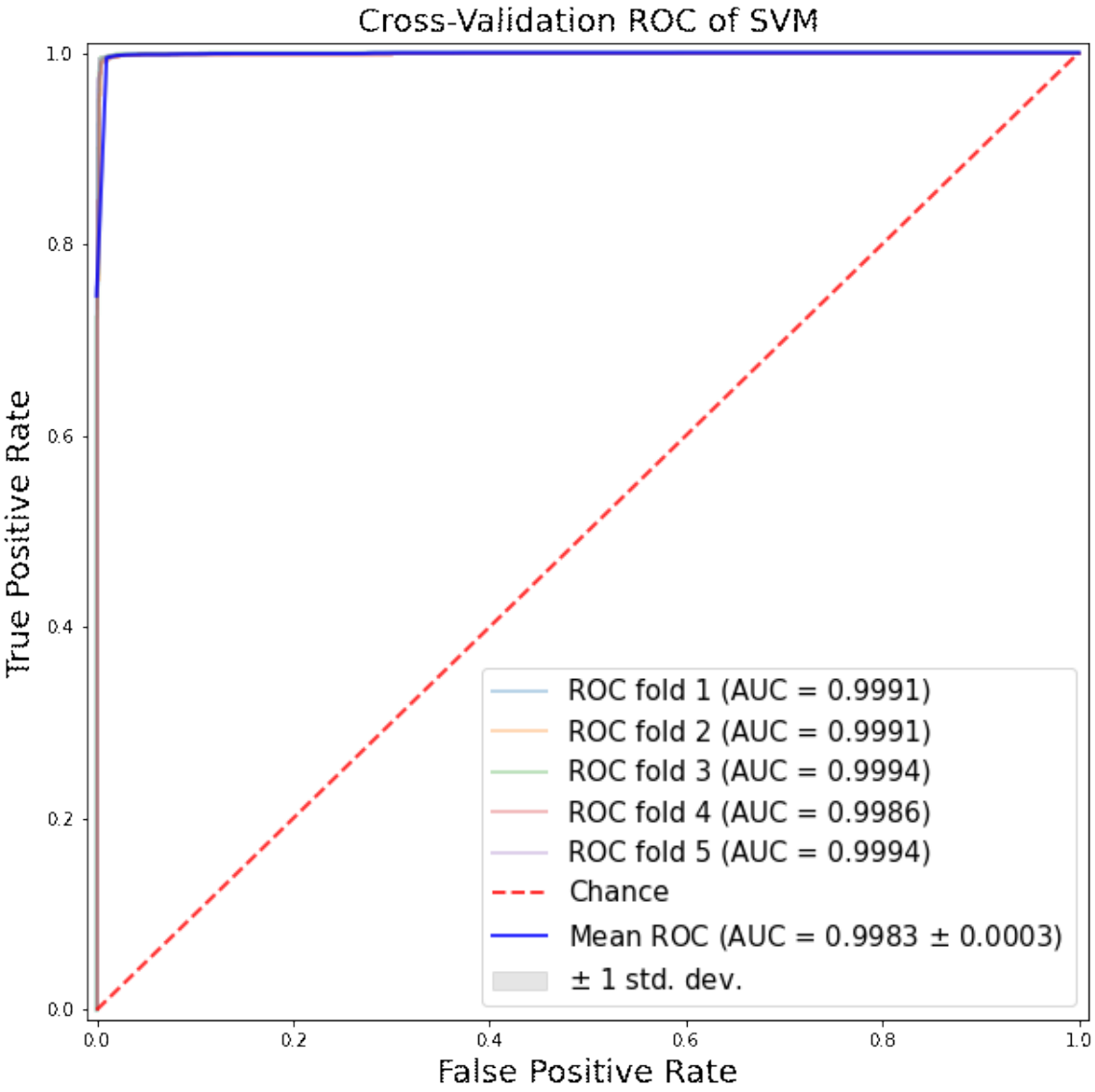

Figura 5.18.: Area Under Curve del Real Life Violence Dataset

5.2. Setting Sperimentale C3D end-to-end

Come detto nel capitolo precedente, nell'architettura end-to-end si è cercato di testare come la scelta dell'optimizer influenzi sulle prestazioni del modello. Per ogni optimizer sono stati testati due valori di epoche.

Per valutare le prestazioni del sistema realizzato con architettura end-to-end, si è deciso di utilizzare come metriche di valutazione l'accuratezza e l'area under the curve (AUC),

sfruttando a tal proposito lo schema della 5-fold cross validation. Come per il modello C3D + SVM, ogni dataset viene diviso in cinque differenti split, per ogni suddivisione vengono stampati come output la matrice di confusione e un rapporto di classificazione. Inoltre, alla fine di ogni test, vengono riportati il valore medio di accuratezza, sensibilità, specificità e punteggio F1, nonché il ROC calcolato in ciascuna suddivisione.

Anche in questo caso, il caricamento in memoria del dataset risultava essere ordinato per classe e quindi è stato necessario utilizzare lo shuffling, tramite la classe StratifiedShuffleSplit definita nella libreria Sklearn.

Il confronto nel Movie Violence Dataset, così come nel caso precedente, non viene riportato in quanto è caratterizzato da un numero estremamente limitato di campioni, ma soprattutto questi risultano molto facili da discriminare.

5.2.1. Test Hockey Fight Dataset

Optimizer Adam 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

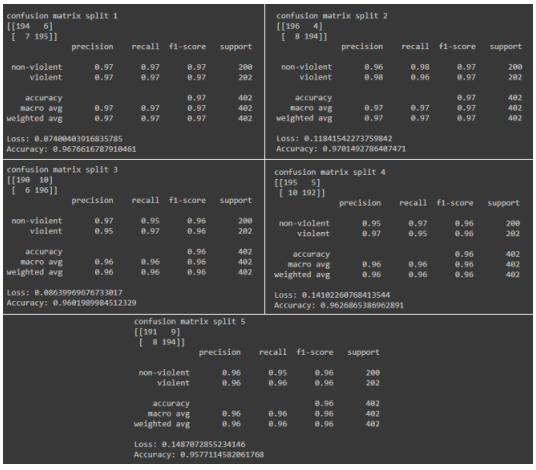

Figura 5.19.: Matrici di confusione sull'Hockey Fight Dataset con optimizer Adam e 20 epoche

Come già visto, la classe 1 rappresenta l'aggressione, mentre la 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.
Di seguito è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

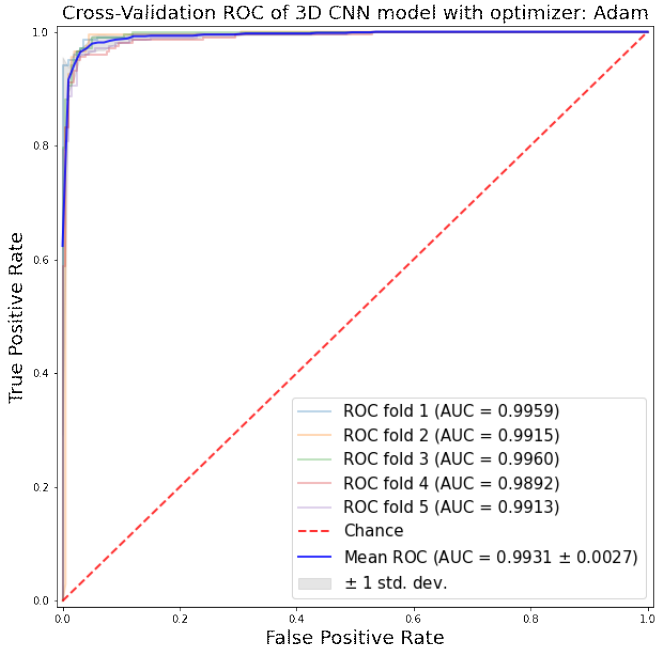

Figura 5.20.: Area Under Curve sull'Hockey Fight Dataset con optimizer Adam e 20 epoche

Come possiamo notare nell'immagine sottostante, i test effettuati su questo dataset hanno prodotto un'accuratezza media del 96,3%, molto vicino all'accuratezza ottenuta con il modello con classificatore SVM.

```
Avg loss: 0.1137098103761673 +/- 0.02937803779594171
Avg accuracy: 0.9636815905570983 +/- 0.004613743550880092
Avg sensitivity: 0.9613861386138612 +/- 0.006567573842287926
Avg specificity: 0.966 +/- 0.011575836902790236
Avg f1-score: 0.963791391120996 +/- 0.004435543540238766
```

Figura 5.21.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con Adam e il numero di epoche pari a 50.

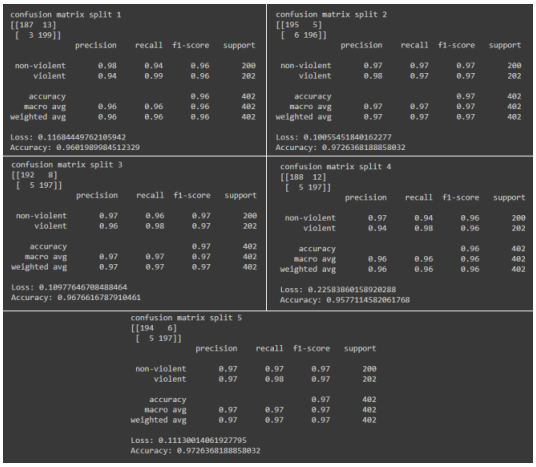

Figura 5.22.: Matrici di confusione sull'Hockey Fight Dataset con optimizer Adam e 50 epoche

L'immagine sottostante raffigura la curva AUC.

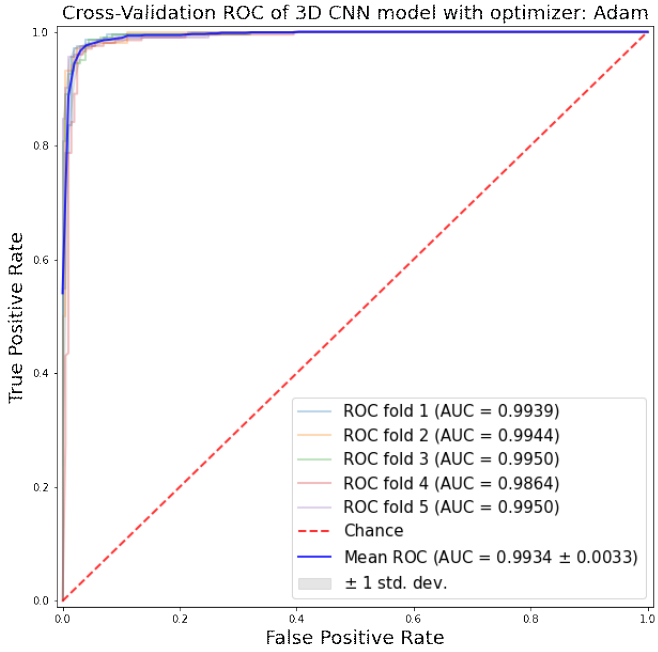

Figura 5.23.: Area Under Curve sull'Hockey Fight Dataset con optimizer Adam e 50 epoche

A differenza del test effettuato con un numero minore di epoche, possiamo notare un piccolo aumento nell'accuracy.

```
Avg loss: 0.13286284506320953 +/- 0.04678212863717183
Avg accuracy: 0.9661691546440124 +/- 0.00621392603463091
Avg sensitivity: 0.9762376237623762 +/- 0.004850474738184516
Avg specificity: 0.9560000000000001 +/- 0.015937377450509212
Avg f1-score: 0.9667186630926423 +/- 0.005817106847313819
```

Figura 5.24.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

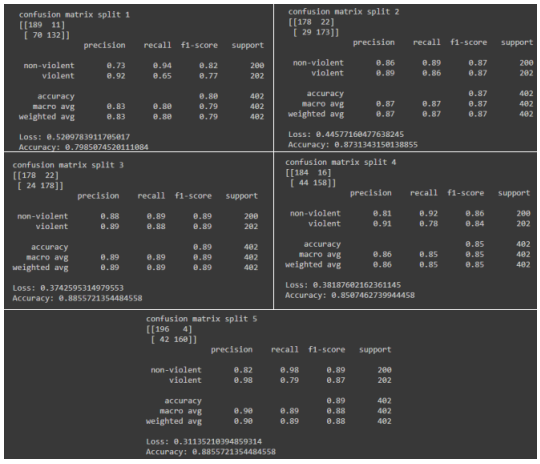

Figura 5.25.: Matrici di confusione sull'Hockey Fight Dataset con optimizer AdaDelta e 20 epoche

Di seguito è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

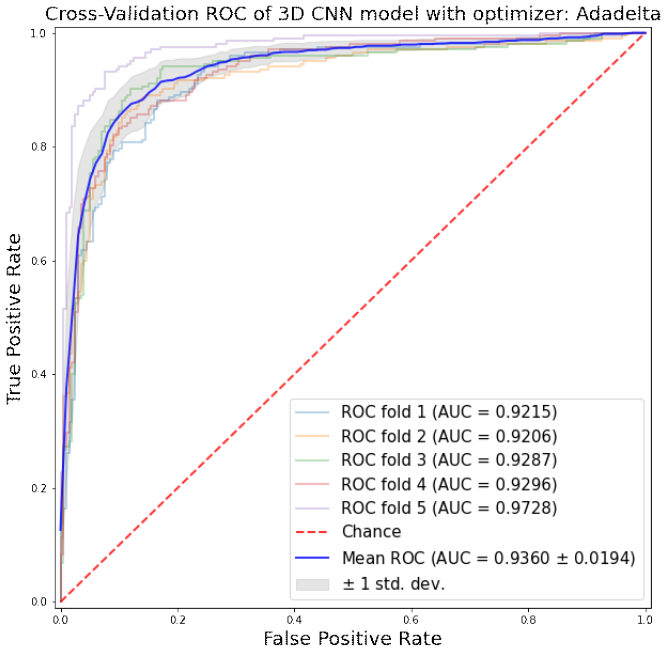

Figura 5.26.: Area Under Curve sull'Hockey Fight Dataset con optimizer AdaDelta e 20 epoche

Si nota come questa curva vari molto da quella prodotta con il precedente optimizer. Anche l'accuratezza ne risente parecchio, infatti, nell'immagine alla pagina successiva, si può notare un drastico calo, in questo test ha un valore intorno all'85,9%.

```
Avg loss: 0.40684753060340884 +/- 0.07119812826531155
Avg accuracy: 0.8587064623832703 +/- 0.032677139284199436
Avg sensitivity: 0.793069306930693 +/- 0.07923266940096062
Avg specificity: 0.925 +/- 0.03435112807463532
Avg f1-score: 0.8474137052685811 +/- 0.04374203721228228
```

Figura 5.27.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con il numero di epoche pari a 50.

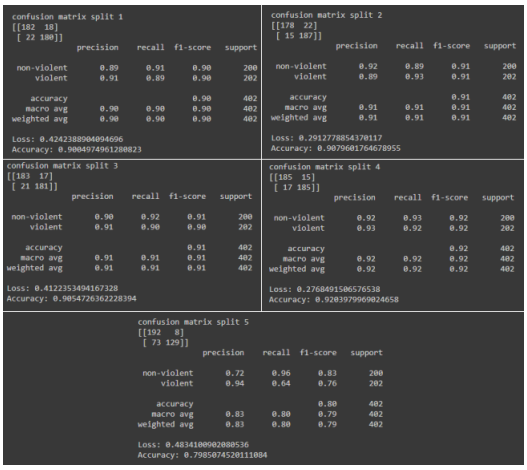

Figura 5.28.: Matrici di confusione sull'Hockey Fight Dataset con optimizer AdaDelta e 50 epoche

Anche in questo caso la curva AUC, come mostrato di seguito, è molto diversa dal caso con optimizer Adam.

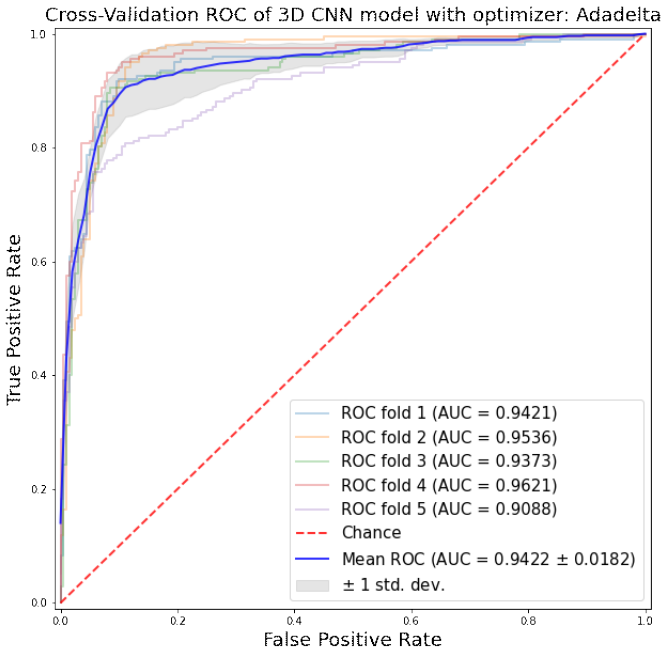

Figura 5.29.: Area Under Curve sull'Hockey Fight Dataset con optimizer AdaDelta e 50 epoche

Nell'immagine finale dei risultati si può vedere come il valore dell'accuratezza sia migliorato, dal test effettuato a 20 epoche, ma ancora non raggiunge i valori ottenuti con gli altri optimizer.

```
Avg loss: 0.3776022732257843 +/- 0.08021572139883075
Avg accuracy: 0.8865671515464782 +/- 0.04451554817377653
Avg sensitivity: 0.8534653465346536 +/- 0.10817029927720923
Avg specificity: 0.9199999999999999 +/- 0.02302172886644266
Avg f1-score: 0.879287125190532 +/- 0.059496654474071944
```

Figura 5.30.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Riportiamo di seguito le 5 matrici di confusione relative all'optimizer scelto per il test e con numero di epoche pari a 20.

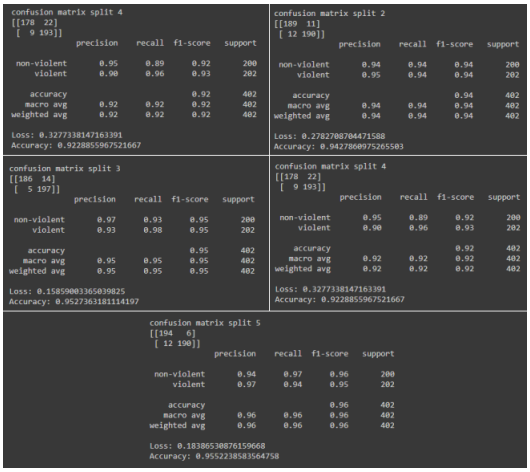

Figura 5.31.: Matrici di confusione sull'Hockey Fight Dataset con optimizer RMSProp e 20 epoche

Di seguito è raffigutata la curva AUC prodotta sempre con il 5-fold cross validation.

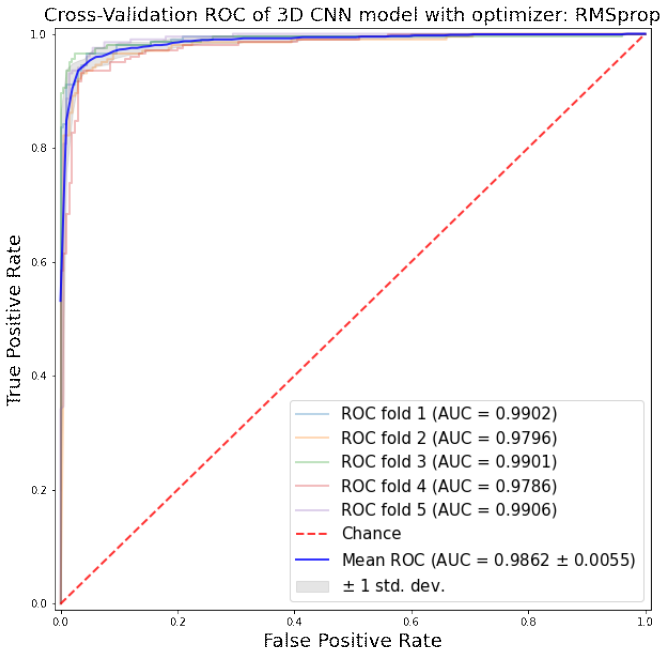

Figura 5.32.: Area Under Curve sull'Hockey Fight Dataset con optimizer RmsProp e 20 epoche

Come possiamo notare nella prossima immagina, i test effettuati su questo dataset hanno prodotto un'accuratezza media pari a 94,5%, anche in questo caso è un valore minore di quello registrato con l'optmizier Adam.

```
Avg loss: 0.22208194434642792 +/- 0.06845017837983587
Avg accuracy: 0.9452736377716064 +/- 0.011981672024846256
Avg sensitivity: 0.9504950495049505 +/- 0.01364757302187152
Avg specificity: 0.9400000000000001 +/- 0.028809720581775847
Avg f1-score: 0.9459474982473939 +/- 0.010998000711602337
```

Figura 5.33.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer RmsProp e 20 epoche

Optimizer RMSProp 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

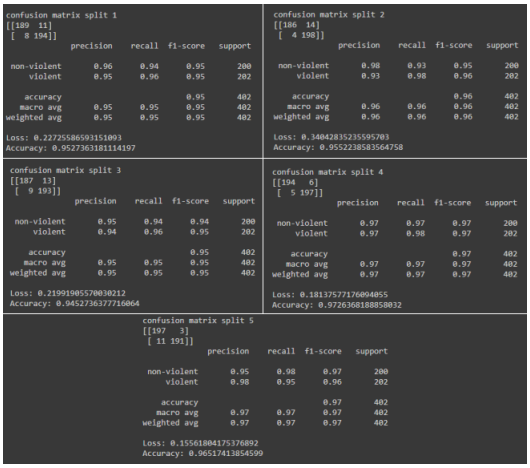

Figura 5.34.: Matrici di confusione sull'Hockey Fight Dataset con optimizer RMSProp e 50 epoche

Di seguito è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

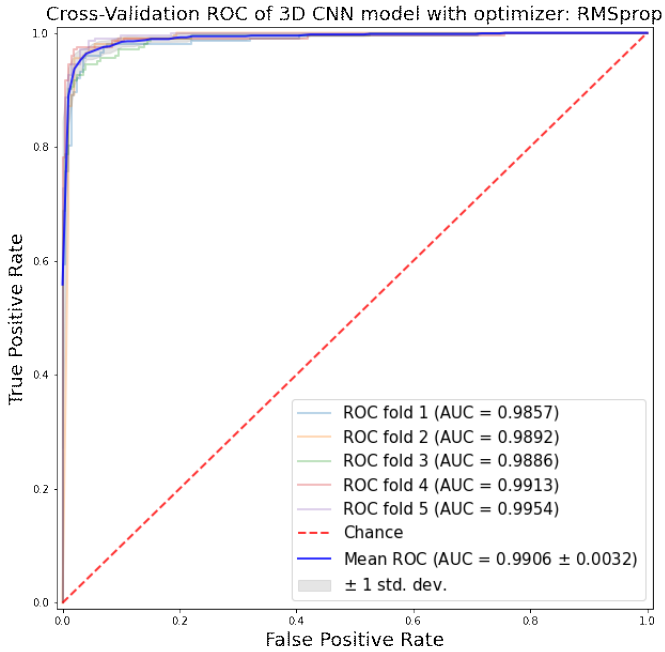

Figura 5.35.: Area Under Curve sull'Hockey Fight Dataset con optimizer RmsProp e 50 epoche

A differenza del test effettuato con 20 epoche, nell'immagine nella pagina successiva possiamo notare un aumento dell'accuracy, passata dal 94% al 95,8%.

```
Avg loss: 0.2249194175004959 +/- 0.0633585138801846
Avg accuracy: 0.958208954334259 +/- 0.009621434679102663
Avg sensitivity: 0.9633663366336634 +/- 0.012756533392797167
Avg specificity: 0.9530000000000001 +/- 0.021118712081942853
Avg f1-score: 0.9586806189278484 +/- 0.009255379031663985
```

Figura 5.36.: Risultati relativi al test sull'Hockey Fight Dataset con optimizer RmsProp e 50 epoche

5.2.2. Test Crowd Violence Dataset

Optimizer Adam 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

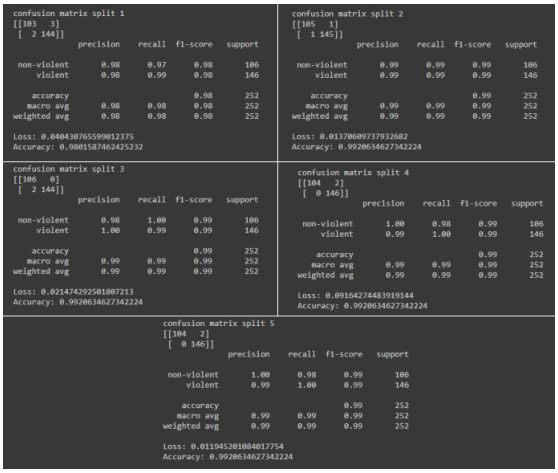

Figura 5.37.: Matrici di confusione sul Crowd Violence Dataset con optimizer Adam e 20 epoche

Come si può vedere, anche se il dataset non risulta perfettamente bilanciato, per via della lunghezza variabile dei filmati, il sistema ha il grande vantaggio di produrre pochi falsi negativi, che ricordiamo essere il caso peggiore. L'immagine di seguito raffigura la curva AUC prodottatramite 5-fold cross validation.

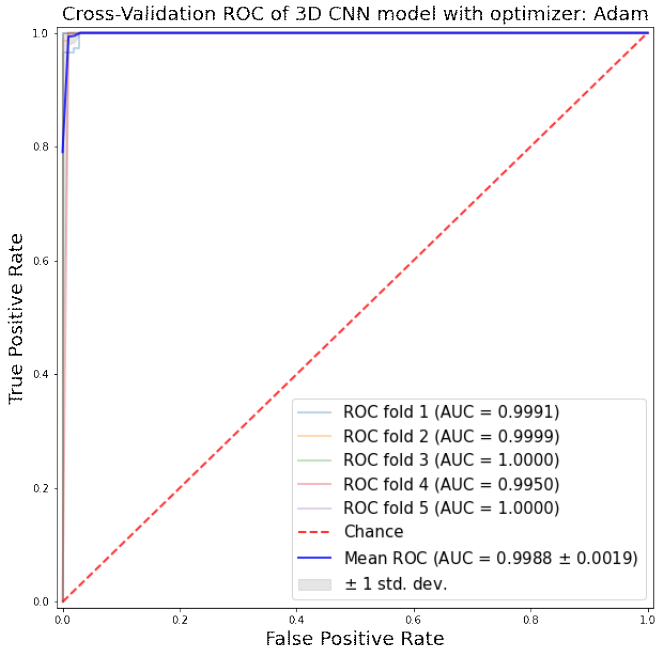

Figura 5.38.: Area Under Curve sul Crowd Violence Dataset con optimizer Adam e 20 epoche

L'accuratezza media di questo optimizer, relativo a questo dataset, è molto elevata ed è pari a 98,9%.

```
Avg loss: 0.03583982028067112 +/- 0.029672431894240785
Avg accuracy: 0.9896825194358826 +/- 0.004761886596679687
Avg sensitivity: 0.9931506849315068 +/- 0.006126213636985749
Avg specificity: 0.9849056603773585 +/- 0.009620791535080723
Avg f1-score: 0.9911167689228233 +/- 0.004090955447506106
```

Figura 5.39.: Risultati relativi al test sul Crowd Violence Dataset con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con Adam e il numero di epoche pari a 50.

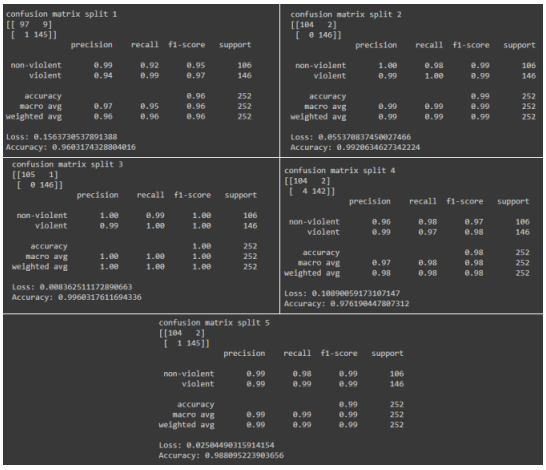

Figura 5.40.: Matrici di confusione sul Crowd Violence Dataset con optimizer Adam e 50 epoche

L'immagine sottostante raffigura la curva AUC.

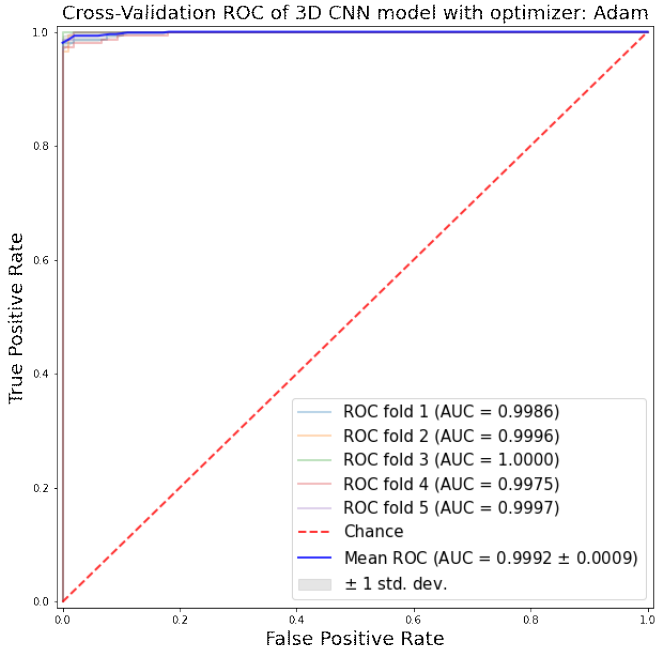

Figura 5.41.: Area Under Curve sul Crowd Violence Dataset con optimizer Adam e 50 epoche

A differenza del test effettuato con un numero minore di epoche, possiamo notare un piccolo decremento nell'accuracy.

```
Avg loss: 0.07081037946045399 +/- 0.05478034326532803
Avg accuracy: 0.9825396656990051 +/- 0.012944062549268285
Avg sensitivity: 0.9917808219178083 +/- 0.010066396203218538
Avg specificity: 0.969811320754717 +/- 0.02760139403458074
Avg f1-score: 0.9851044826545424 +/- 0.010886242371114827
```

Figura 5.42.: Risultati relativi al test sul Crowd Violence Dataset con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

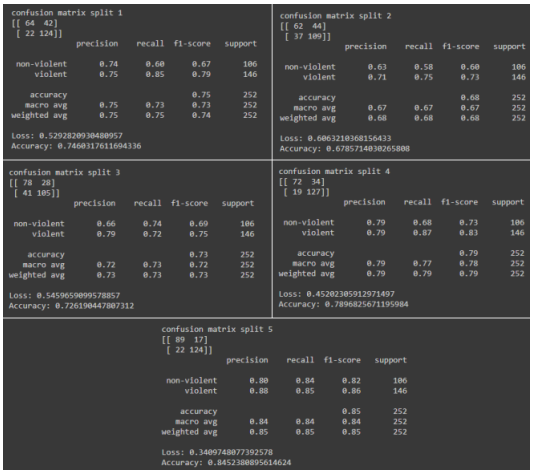

Figura 5.43.: Matrici di confusione sul Crowd Violence Dataset con optimizer AdaDelta e 20 epoche

Come possiamo notare, anche con questo dataset, l'optmizier AdaDelta ha dei risultati abbastanza bassi. Ci sono un elevato numero di errori di tipo 1 e di tipo 2. Lo si può vedere anche nella curva AUC, raffigurata nella prossima immagine, come questo optimizier ha un'accuratezza abbastanza bassa.

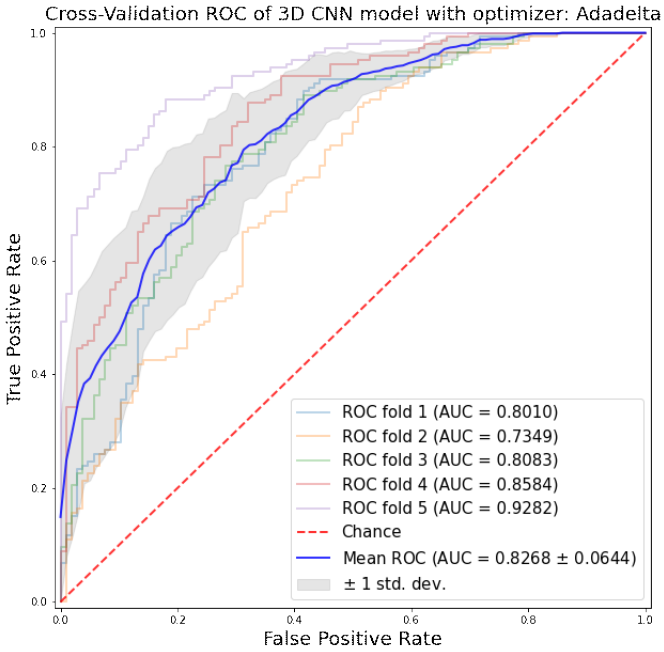

Figura 5.44.: Area Under Curve sul Crowd Violence Dataset con optimizer AdaDelta e 20 epoche

I risultati ottenuti su questo dataset sono nettamente inferiori a quelli ottenuti con gli altri optimizier, infatti l'accuratezza media è solo del 75%.

```
Avg loss: 0.4949133813381195 +/- 0.09136290155513567
Avg accuracy: 0.7571428537368774 +/- 0.056700236683122136
Avg sensitivity: 0.8068493150684931 +/- 0.06147617992415583
Avg specificity: 0.6886792452830188 +/- 0.0928179160471611
Avg f1-score: 0.7936260037314232 +/- 0.04888301260994914
```

Figura 5.45.: Risultati relativi al test sul Crowd Violence Dataset con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con il numero di epoche pari a 50.

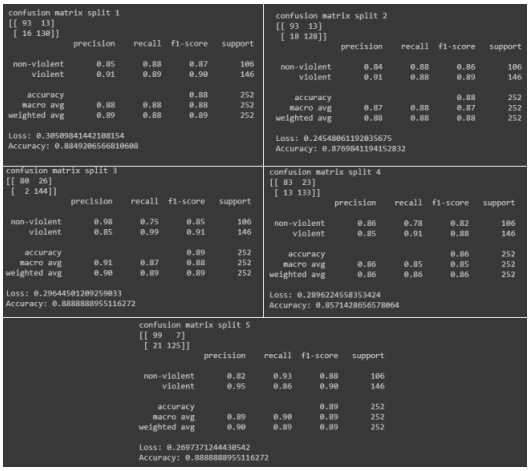

Figura 5.46.: Matrici di confusione sul Crowd Violence Dataset con optimizer AdaDelta e 50 epoche

Dal test effettuato con un numero minore di epoche, si nota subito un miglioramento, anche nel numero di errori di tipo 1 e di tipo 2, nettamente inferiori al caso precedente. Anche nella curva AUC, si nota un miglioramento.

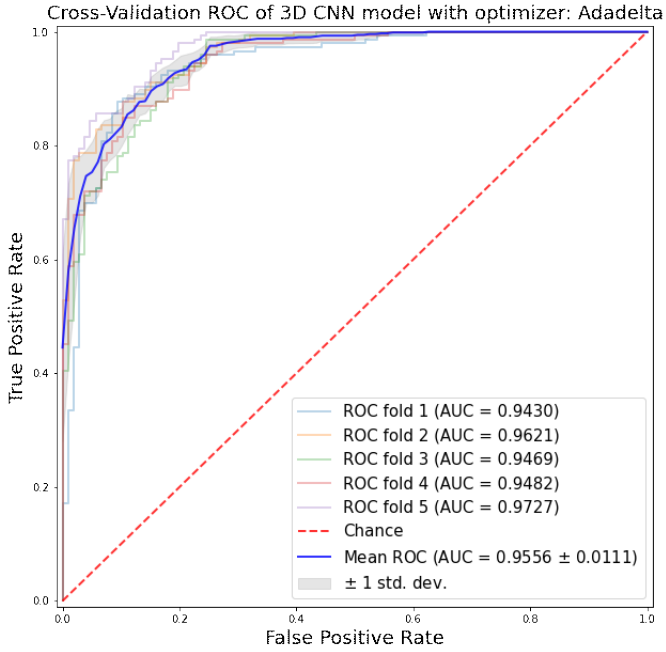

Figura 5.47.: Area Under Curve sul Crowd Violence Dataset con optimizer AdaDelta e 50 epoche

Nell'ultima immagine, quella riportante i risultati, abbiamo un aumento notevole dell'accuratezza, dovuto al numero di epoche, rispetto al test effettuato precedentemente.

```
Avg loss: 0.281276723742485 +/- 0.02136223366597303
Avg accuracy: 0.879365086555481 +/- 0.011931188873153616
Avg sensitivity: 0.9041095890410957 +/- 0.044809389681633206
Avg specificity: 0.8452830188679246 +/- 0.06633356917451676
Avg f1-score: 0.8966215449092276 +/- 0.010064005798754537
```

Figura 5.48.: Risultati relativi al test sul Crowd Violence Dataset con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

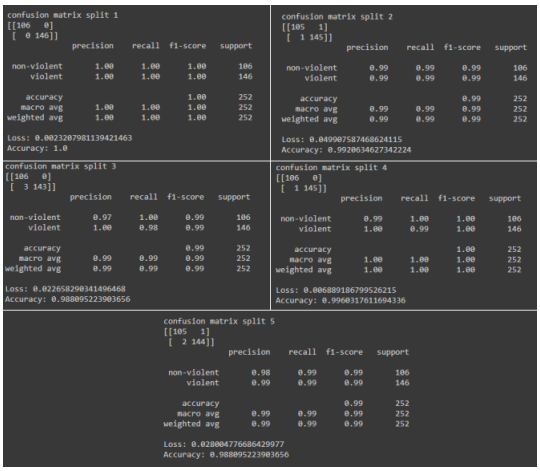

Figura 5.49.: Matrici di confusione sul Crowd Violence Dataset con optimizer RMSProp e 20 epoche

Di seguito riportiamo la curva AUC.

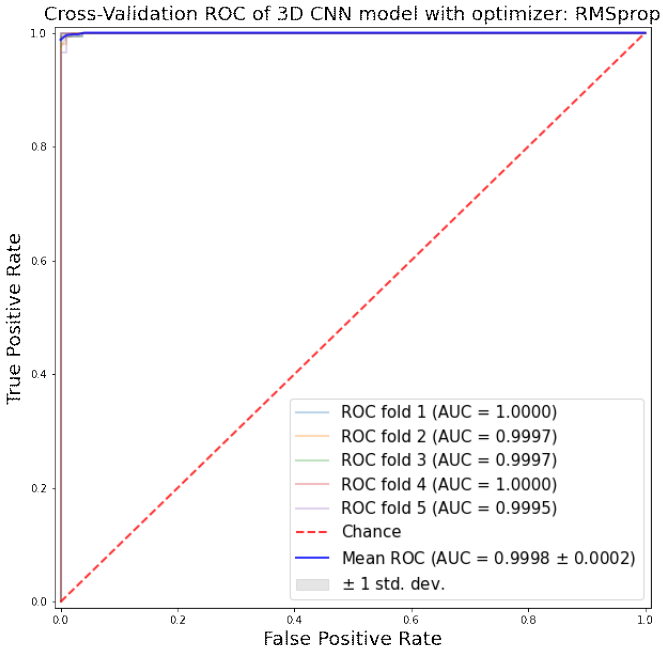

Figura 5.50.: Area Under Curve sul Crowd Violence Dataset con optimizer RMSProp e 20 epoche

Come possiamo notare nell'immagine sottostante riguardante i risultati, questo optimizier raggiunge un'accuratezza media del 99%.

```
Avg loss: 0.021956127882003785 +/- 0.01691698652943044
Avg accuracy: 0.9928571343421936 +/- 0.004627748523165625
Avg sensitivity: 0.9904109589041095 +/- 0.006984958237798327
Avg specificity: 0.9962264150943396 +/- 0.004621678759968259
Avg f1-score: 0.9938048715253565 +/- 0.0040215073760326734
```

Figura 5.51.: Risultati relativi al test sul Crowd Violence Dataset con optimizer RMSProp e 20 epoche

Optimizer RMSProp 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con RMSProp il numero di epoche pari a 50.

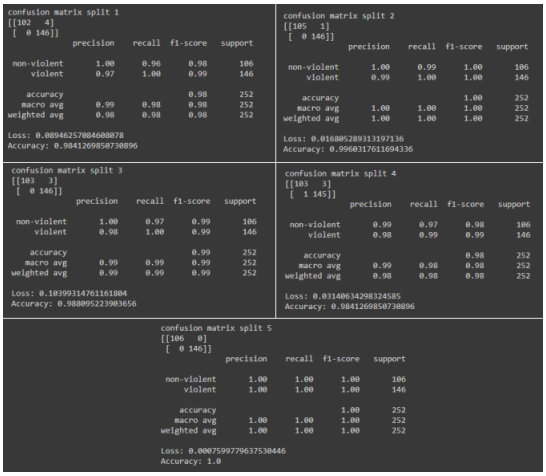

Figura 5.52.: Matrici di confusione sul Crowd Violence Dataset con optimizer RMSProp e 50 epoche

Nell'immagine sottostante è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

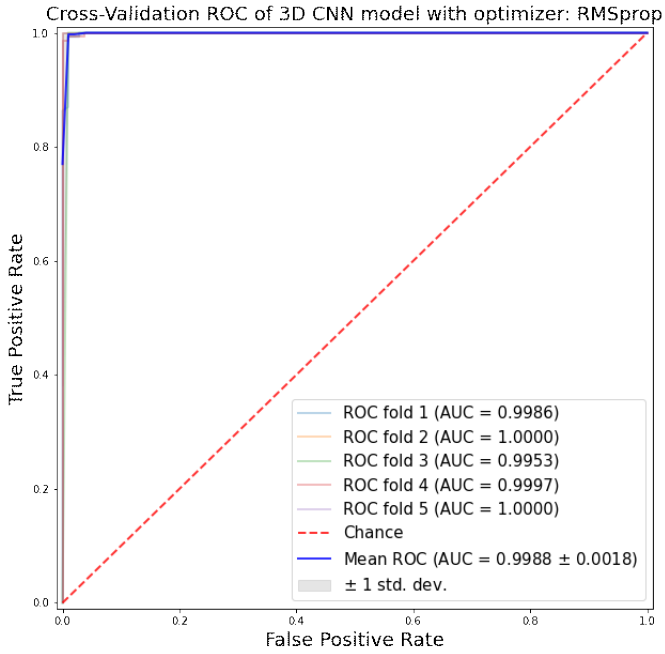

Figura 5.53.: Area Under Curve sul Crowd Violence Dataset con optimizer RMSProp e 50 epoche

Con un numero di epoche maggiore, non abbiamo un valore migliore per quanto riguarda l'accuratezza media, come si può notare nell'immagine alla pagina successiva.

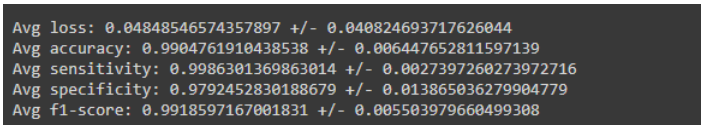

Figura 5.54.: Risultati relativi al test sul Crowd Violence Dataset con optimizer RMSProp e 50 epoche

5.2.3. Test Dataset Intero

Anche per questo modello si è voluto valutare la capacità di generalizzazione del sistema realizzato. Ricordiamo che questo dataset è una base di dati formata dall'unione dei tre dataset, Hockey Fight Dataset, Crowd Violence Dataset e Movie Violence Dataset.

Optimizer Adam 20 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con Adam e numero di epoche pari a 20.

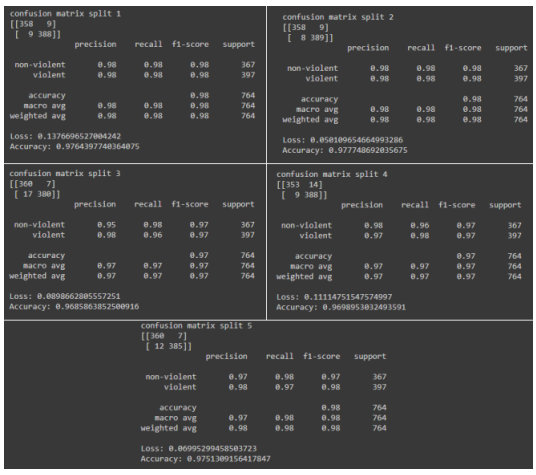

Figura 5.55.: Matrici di confusione sul Dataset Intero con optimizer Adam e 20 epoche

Nell'immagine sottostante viene riportata la curva AUC.

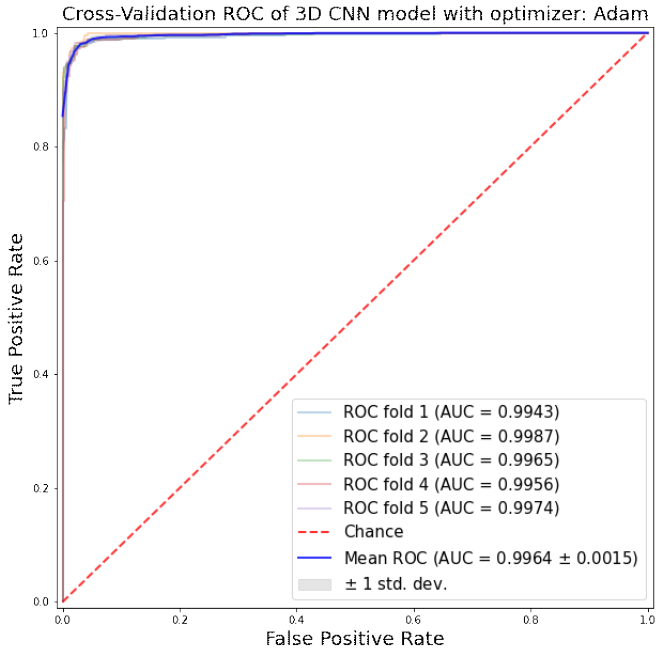

Figura 5.56.: Area Under Curve sul Dataset Intero con optimizer Adam e 20 epoche

Come si nota nell'ultima immagine relativa ai risultati ottenuti in questo test, il modello con l'optimizer Adam ha ottenuto un'accuratezza media pari a 97,3%.

```
Avg loss: 0.09174921959638596 +/- 0.030651115187991632
Avg accuracy: 0.9735602140426636 +/- 0.003646172271017682
Avg sensitivity: 0.9722921914357683 +/- 0.008277922783453387
Avg specificity: 0.9749318801089919 +/- 0.006978882002651626
Avg f1-score: 0.9744933968600833 +/- 0.0035748806218408083
```

Figura 5.57.: Risultati relativi al test sul Dataset Intero con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

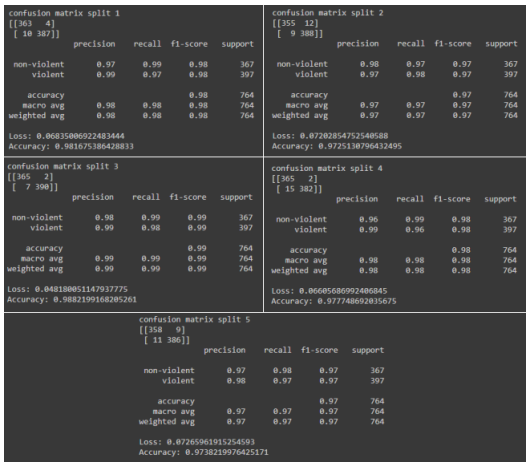

Figura 5.58.: Matrici di confusione sul Dataset Intero con optimizer Adam e 50 epoche

Di seguito riportiamo la curva AUC prodotta sempre tramite 5-fold cross validation.

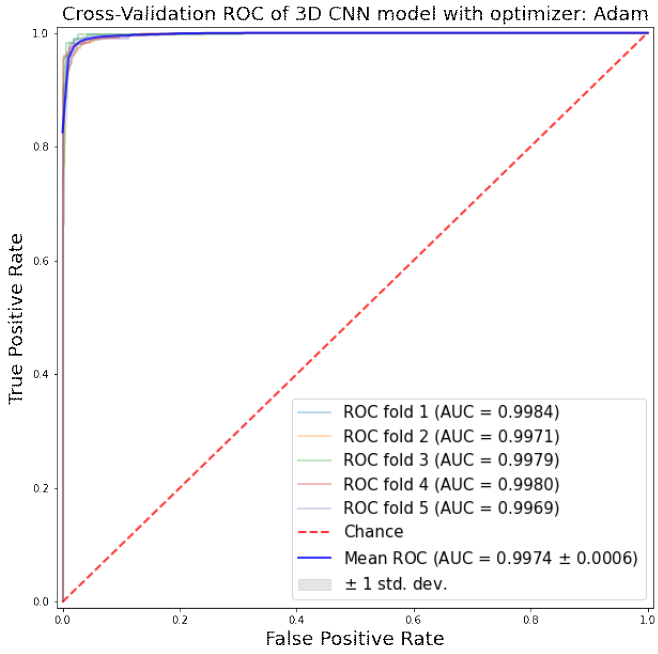

Figura 5.59.: Area Under Curve sul Dataset Intero con optimizer Adam e 50 epoche

A differenza del test effettuato con un numero minore di epoche, possiamo notare un piccolo decremento nell'accuracy.

```
Avg loss: 0.0654550313949585 +/- 0.008969789242345896
Avg accuracy: 0.9787958145141602 +/- 0.005699363401440053
Avg sensitivity: 0.9738035264483628 +/- 0.00668337489240381
Avg specificity: 0.9841961852861036 +/- 0.010953542911303446
Avg f1-score: 0.9794916719440063 +/- 0.005450824277012381
```

Figura 5.60.: Risultati relativi al test sul Dataset Intero con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine possiamo vedere le 5 matrici di confusione relative al test con AdaDelta.

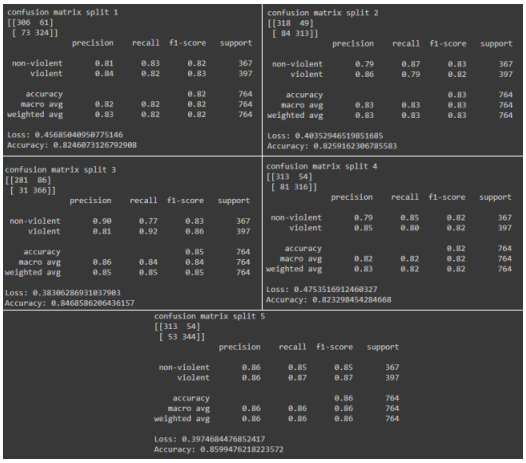

Figura 5.61.: Matrici di confusione sul Dataset Intero con optimizer AdaDelta e 20 epoche

Anche su questo dataset, notiamo un incremento degli errori di tipo 1 e tipo 2. Riportiamo di seguito la curva AUC.

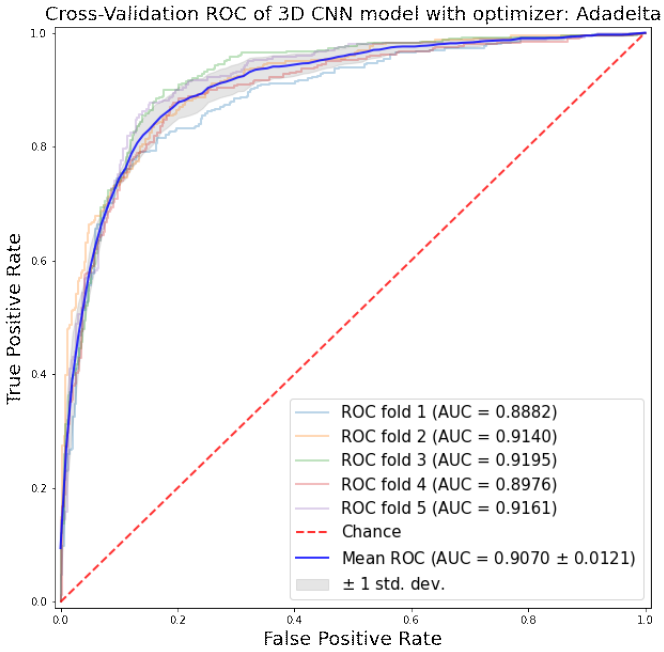

Figura 5.62.: Area Under Curve sul Dataset Intero con optimizer AdaDelta e 20 epoche

I risultati ottenuti, come mostrati nell'immagine sottostante, non sono ottimi, l'accuratezza media è uguale a 83,6%.

```
Avg loss: 0.4232525765895844 +/- 0.036089242503192696
Avg accuracy: 0.836125648021698 +/- 0.014724973940730614
Avg sensitivity: 0.8377833753148615 +/- 0.050115184344366286
Avg specificity: 0.8343324250681199 +/- 0.03587636096050746
Avg f1-score: 0.8410006244496868 +/- 0.018709808554822627
```

Figura 5.63.: Risultati relativi al test sul Dataset Intero con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Riportiamo di seguito le 5 matrici di confusione, per il test effettuato con numero di epoche uguale a 50.

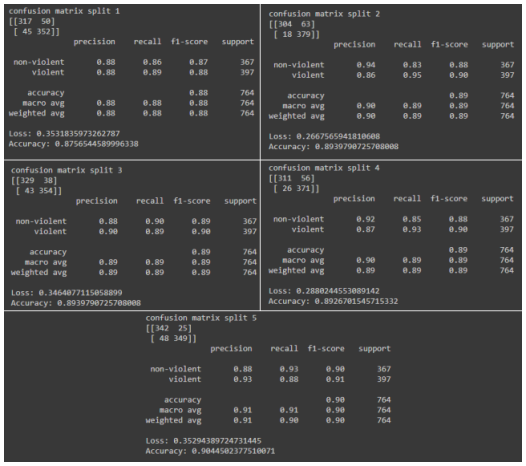

Figura 5.64.: Matrici di confusione sul Dataset Intero con optimizer AdaDelta e 50 epoche

Anche qui notiamo un elevato numero di errori di tipo 1 e tipo 2. Nell'immagine sottostante è raffigurata la curva AUC.

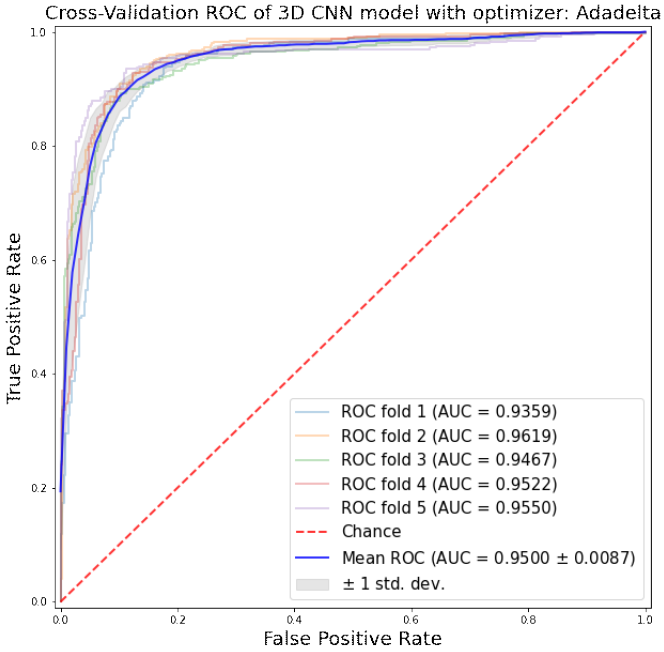

Figura 5.65.: Area Under Curve sul Dataset Intero con optimizer AdaDelta e 50 epoche

Possiamo notare un aumento rispetto al test con 20 epoche, infatti, l'accuratezza media è pari all' 89%. I risultati, comunque, sono molto bassi rispetto agli altri due optimizer.

```
Avg loss: 0.3214632511138916 +/- 0.03668898724879956
Avg accuracy: 0.8921465992927551 +/- 0.009277502223322957
Avg sensitivity: 0.9093198992443325 +/- 0.0297613208820996
Avg specificity: 0.8735694822888282 +/- 0.036743341321062835
Avg f1-score: 0.8975398963230592 +/- 0.008653822168567868
```

Figura 5.66.: Risultati relativi al test sul Dataset Intero con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split.

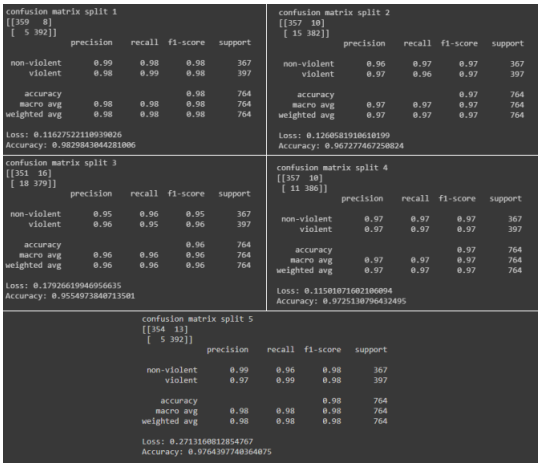

Figura 5.67.: Matrici di confusione sul Dataset Intero con optimizer RMSProp e 20 epoche

Di seguito riportiamo la curva AUC prodotta tramite 5-fold cross validation.

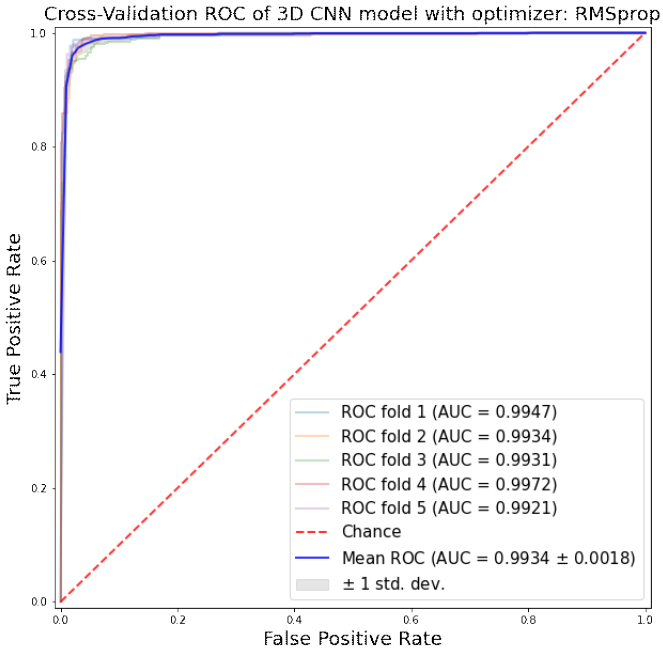

Figura 5.68.: Area Under Curve sul Dataset Intero con optimizer RMSProp e 20 epoche

Anche con questo optimizier, come possiamo notare dall'immagine, l'accuratezza media è del 97%.

```
Avg loss: 0.16158528178930281 +/- 0.05972910372510048
Avg accuracy: 0.9709424018859864 +/- 0.00927012262255045
Avg sensitivity: 0.9727959697732997 +/- 0.01317551317946804
Avg specificity: 0.9689373297002726 +/- 0.007629427792915531
Avg f1-score: 0.9720296513681586 +/- 0.009014077641375504
```

Figura 5.69.: Risultati relativi al test sul Dataset Intero con optimizer RMSProp e 20 epoche

Optimizer RMSProp 50 Epoche

Nell'immagine nella pagina successiva sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

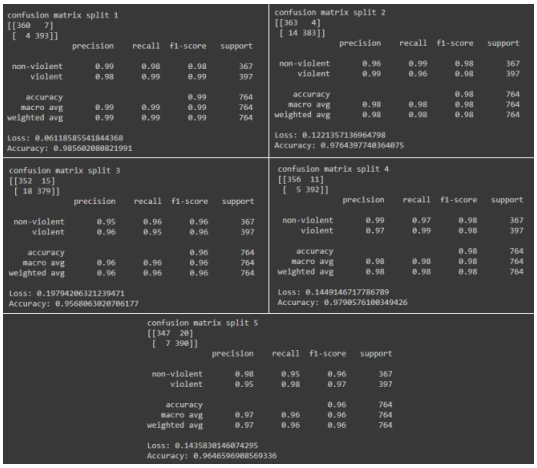

Figura 5.70.: Matrici di confusione sul Dataset Intero con optimizer RMSProp e 50 epoche

Nell'immagine sottostante è raffigurata la curva AUC.

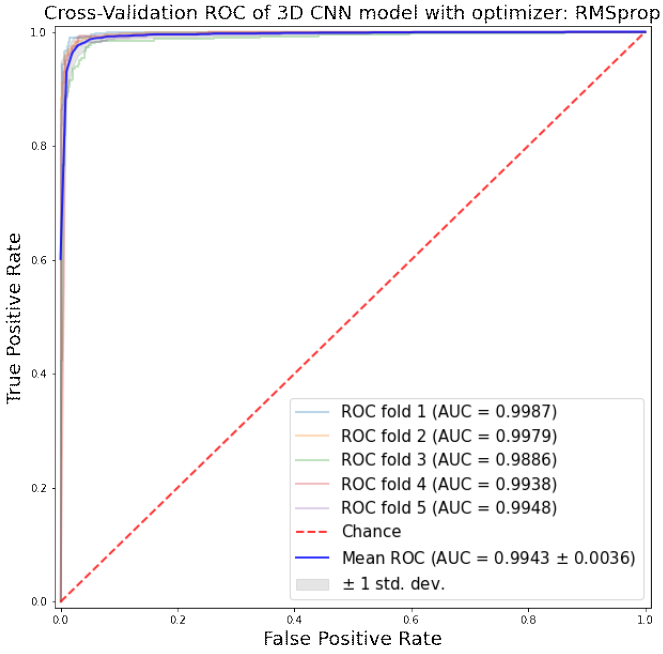

Figura 5.71.: Area Under Curve sul Dataset Intero con optimizer RMSProp e 50 epoche

Come si nota nell'immagine sottostante relativa ai risultati, l'accuratezza media è poco maggiore al caso con il numero di epoche pari a 20.

```
Avg loss: 0.133952263742685 32 +/- 0.044141140048133 67
Avg accuracy: 0.9725130915641784 +/- 0.010372563192447667
Avg sensitivity: 0.9758186397984886 +/- 0.013759698018889337
Avg specificity: 0.9689373297002725 +/- 0.01547146546766038
Avg f1-score: 0.9736124936128796 +/- 0.009959931685528986
```

Figura 5.72.: Risultati relativi al test sul Dataset Intero con optimizer RMSProp e 50 epoche

5.2.4. Test AIRTLab Violence Dataset

Come detto nel paragrafo precedente, questo dataset è stato realizzato all'interno dell'AIRTLab dell'Università Politecninca delle Marche. Ricordiamo che questo dataset non è bilanciato, e i filmati hanno lunghezza variabile.

Optimizer Adam 20 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con l'optimizer Adam e il numero di epoche pari a 20.

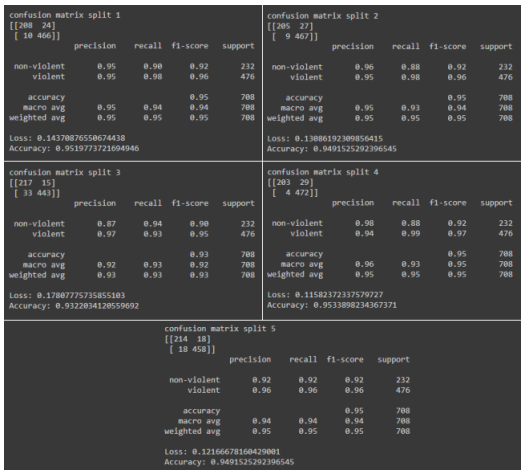

Figura 5.73.: Matrici di confusione sull'AIRTLab Dataset con optimizer Adam e 20 epoche

Il numero di errori di tipo 1 e di tipo 2 sono abbastanza elevati, dovuto alla presenza nei video non violenti di comportamenti (ad esempio abbracci) che possono essere scambiati per violenti. Nell'immagine sottostante riportiamo anche la curva AUC prodotta come sempre tramite 5-fold cross validation.

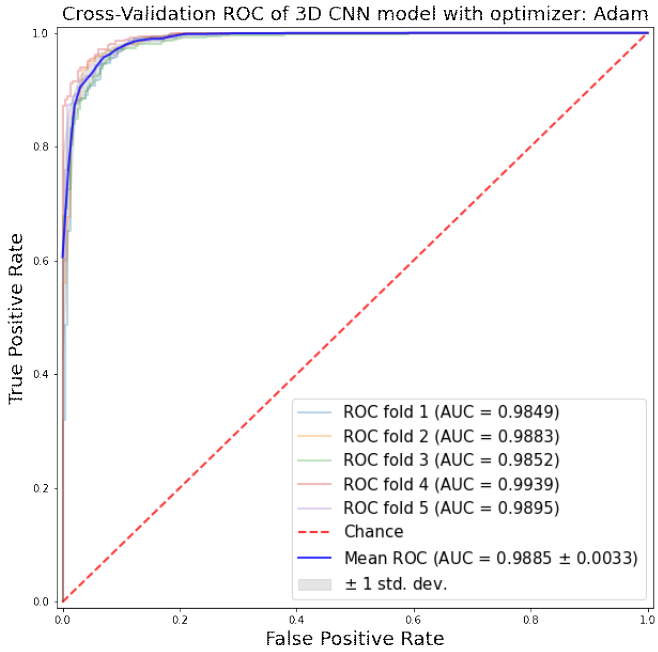

Figura 5.74.: Area Under Curve sull'AIRTLab Dataset con optimizer Adam e 20 epoche

Proprio per la presenza di questi video che causano falsi positivi, l'accuratezza media è pari al 94,7%.

```
Avg loss: 0.13802779018878936 +/- 0.022128380812679687
Avg accuracy: 0.947175133228302 +/- 0.007663634882594922
Avg sensitivity: 0.96890756302521 +/- 0.021317060325992968
Avg specificity: 0.9025862068965518 +/- 0.022905741819976583
Avg f1-score: 0.960941210719787 +/- 0.00632851419982377
```

Figura 5.75.: Risultati relativi al test sull'AIRTLab Dataset con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Nell'immagine nella pagina successiva sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

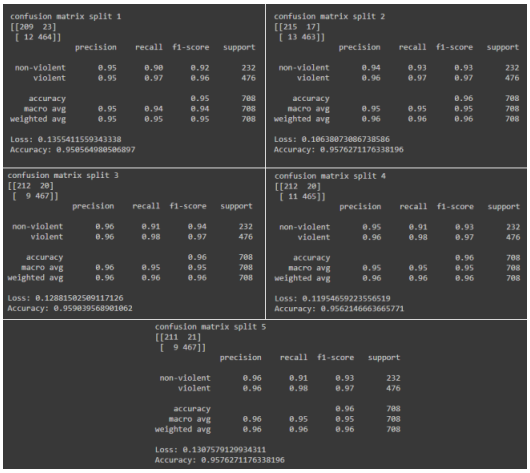

Figura 5.76.: Matrici di confusione sull'AIRTLab Dataset con optimizer Adam e 50 epoche

Di seguito è raffigurata la curva AUC.

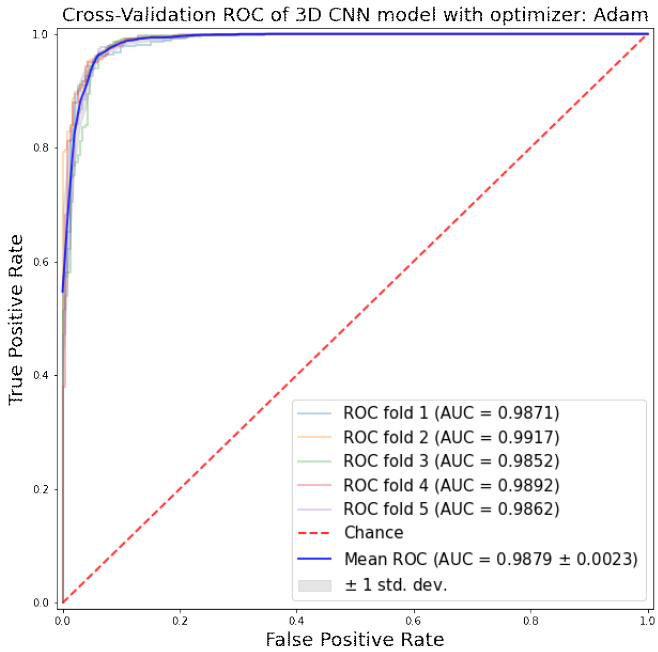

Figura 5.77.: Area Under Curve sull'AIRTLab Dataset con optimizer Adam e 50 epoche

A differenza del test effettuato con un numero minore di epoche, possiamo notare un piccolo aumento nell'accuracy.

```
Avg loss: 0.12420828342437744 +/- 0.010316072028785641
Avg accuracy: 0.9562146902084351 +/- 0.002962737308781744
Avg sensitivity: 0.9773109243697478 +/- 0.003361344537815114
Avg specificity: 0.9129310344827586 +/- 0.00835806871968333
Avg f1-score: 0.9677563736097511 +/- 0.0021614882542538238
```

Figura 5.78.: Risultati relativi al test sull'AIRTLab Dataset con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 20.

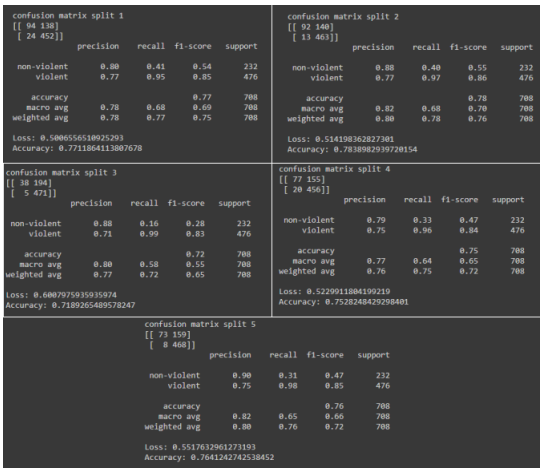

Figura 5.79.: Matrici di confusione sull'AIRTLab Dataset con optimizer AdaDelta e 20 epoche

Di seguito è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

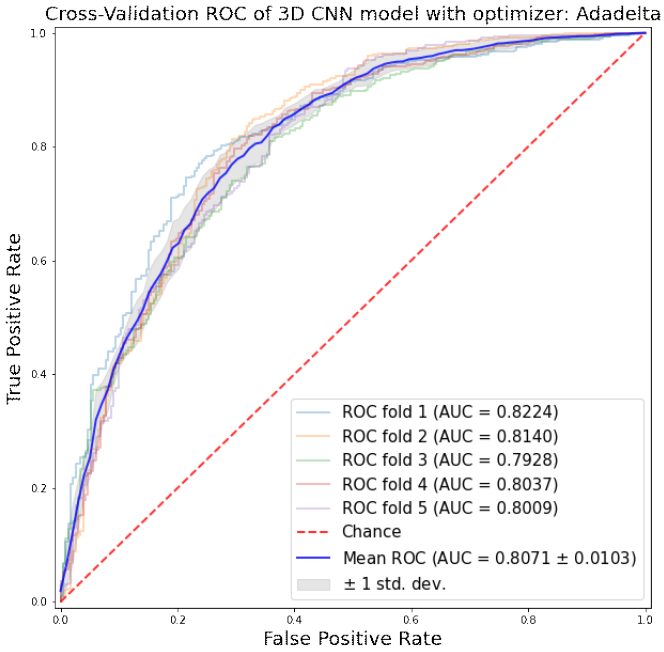

Figura 5.80.: Area Under Curve sull'AIRTLab Dataset con optimizer AdaDelta e 20 epoche

Si nota come questa curva vari molto da quella prodotta con il precedente optimizer. Anche l'accuratezza ne risente parecchio, infatti, si può notare un drastico calo, in questo test ha un valore intorno all'75,8%.

```
Avg loss: 0.5380812168121338 +/- 0.0355508539790343
Avg accuracy: 0.7581920742988586 +/- 0.02207008008714408
Avg sensitivity: 0.9705882352941175 +/- 0.014973554353954175
Avg specificity: 0.32241379310344825 +/- 0.08678256070118545
Avg f1-score: 0.8438849650741261 +/- 0.01097988874392925
```

Figura 5.81.: Risultati relativi al test sull'AIRTLab Dataset con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con il numero di epoche pari a 50.

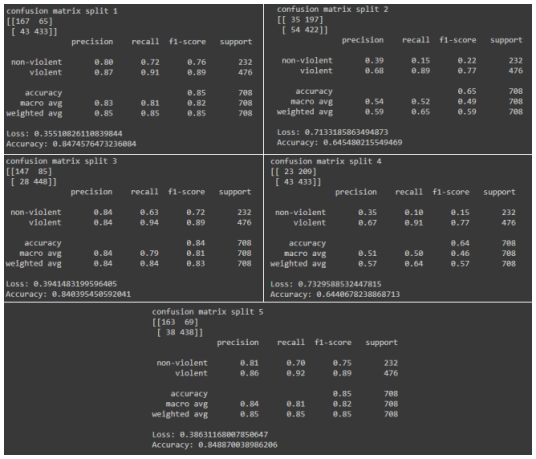

Figura 5.82.: Matrici di confusione sull'AIRTLab Dataset con optimizer AdaDelta e 50 epoche

Anche in questo caso la curva AUC, come mostrato di seguito, è molto diversa dal caso con optimizer Adam.

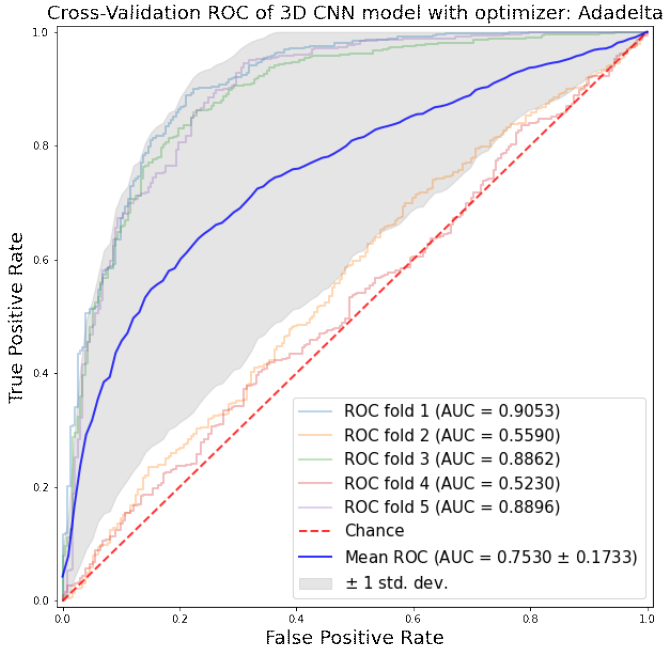

Figura 5.83.: Area Under Curve sull'AIRTLab Dataset con optimizer AdaDelta e 50 epoche

Nell'immagine finale dei risultati si può vedere come il valore dell'accuratezza sia migliorato, dal test effettuato a 20 epoche, ma ancora non raggiunge i valori ottenuti con gli altri optimizer.

```
Avg loss: 0.5163691401481628 +/- 0.16944505368660773
Avg accuracy: 0.7652542352676391 +/- 0.09841460047002254
Avg sensitivity: 0.9134453781512605 +/- 0.01769700814172495
Avg specificity: 0.4612068965517241 +/- 0.27650786478247524
Avg f1-score: 0.8427296530433124 +/- 0.05721132779958888
```

Figura 5.84.: Risultati relativi al test sull'AIRTLab Dataset con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Riportiamo di seguito le 5 matrici di confusione relative all'optimizer scelto per il test econ numero di epoche pari a 20.

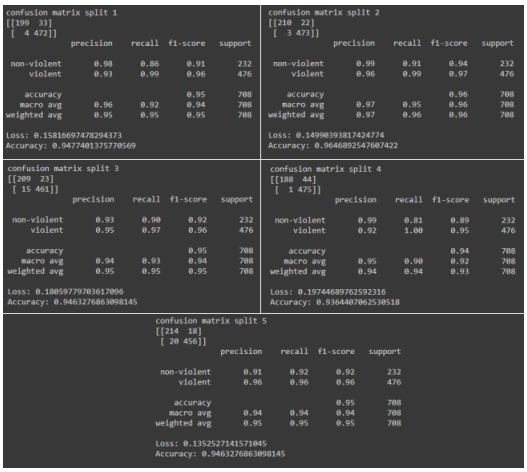

Figura 5.85.: Matrici di confusione sull'AIRTLab Dataset con optimizer RMSProp e 20 epoche

Di seguito è raffigutata la curva AUC prodotta sempre con il 5-fold cross validation.

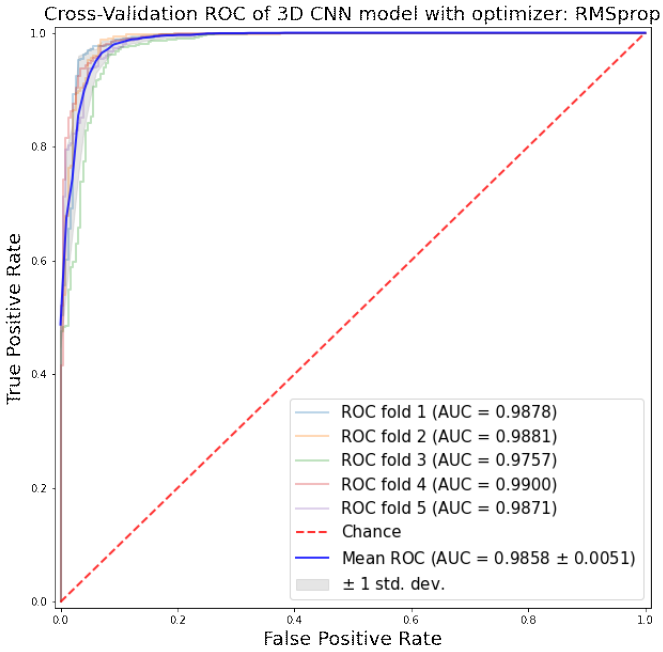

Figura 5.86.: Area Under Curve sull'AIRTLab Dataset con optimizer RMSProp e 20 epoche

Come possiamo notare nella prossima immagine, i test effettuati su questo dataset hanno prodotto un'accuratezza media pari a 94,8%, molto simile al valore ottenuto con l'optmizier Adam.

```
Avg loss: 0.16427366435527802 +/- 0.022149207443461924
Avg accuracy: 0.948305094242096 +/- 0.00913612476436277 5
Avg sensitivity: 0.9819327731092438 +/- 0.015754901898531657
Avg specificity: 0.8793103448275861 +/- 0.04052641128807987
Avg f1-score: 0.9623454534760031 +/- 0.006455262025619371
```

Figura 5.87.: Risultati relativi al test sull'AIRTLab Dataset con optimizer RMSProp e 20 epoche

Optimizer RMSProp 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

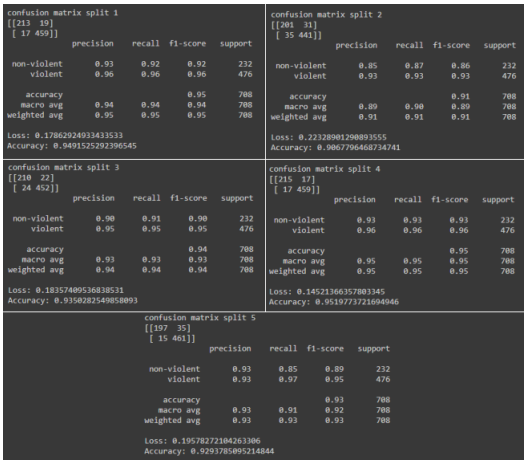

Figura 5.88.: Matrici di confusione sull'AIRTLab Dataset con optimizer RMSProp e 50 epoche

Di seguito è raffigurata la curva AUC prodotta come sempre tramite 5-fold cross validation.

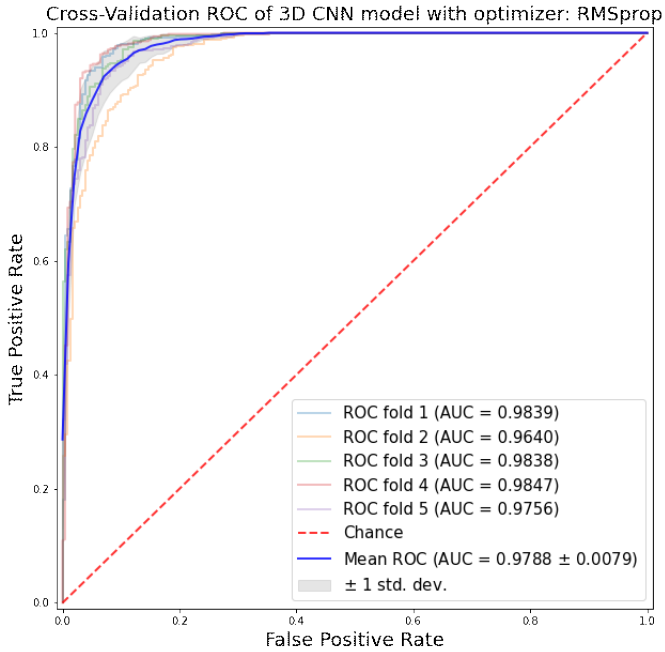

Figura 5.89.: Area Under Curve sull'AIRTLab Dataset con optimizer RMSProp e 50 epoche

A differenza del test effettuato con 20 epoche, notiamo un decremento dell'accuracy, passata dal 94,8% al 93,4%.

```
Avg loss: 0.18529774844646454 +/- 0.025327631337657468
Avg accuracy: 0.9344632625579834 +/- 0.016217737810080533
Avg sensitivity: 0.9546218487394957 +/- 0.015472229108841983
Avg specificity: 0.8931034482758621 +/- 0.030160096007820512
Avg f1-score: 0.9514136460429736 +/- 0.01211627841072895
```

Figura 5.90.: Risultati relativi al test sull'AIRTLab Dataset con optimizer RMSProp e 50 epoche

5.2.5. Test Surveillance Camera Fight Dataset

Ricordiamo che questo dataset è fortemente bilanciato e presenta video di lunghezza media pari a 2 secondi.

Optimizer Adam 20 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con Adam e numero di epoche pari a 20.

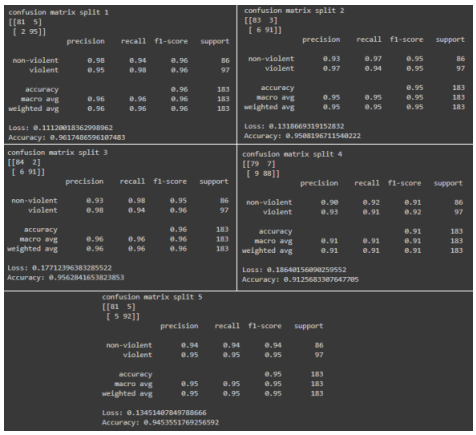

Figura 5.91.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer Adam e 20 epoche

Nell'immagine sottostante viene riportata la curva AUC.

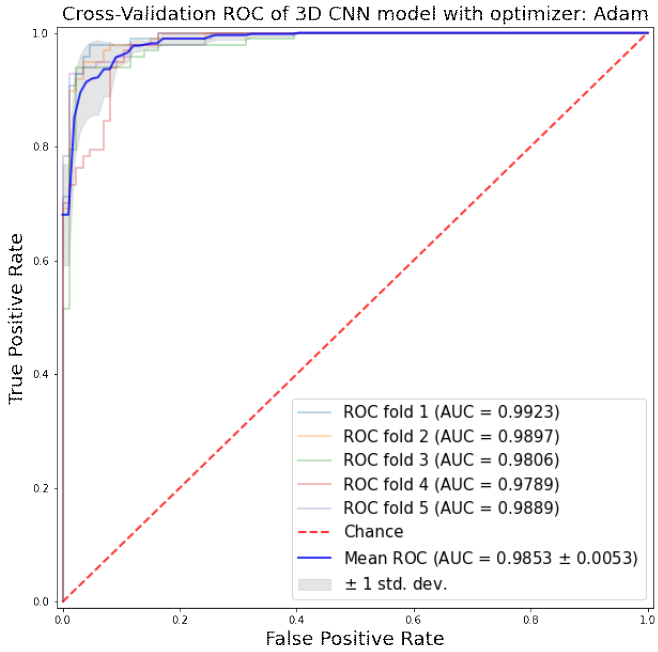

Figura 5.92.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer Adam e 20 epoche

Come si nota nell'ultima immagine relativa ai risultati ottenuti in questo test, il modello con l'optimizer Adam ha ottenuto un'accuratezza media pari a 94,3%.

```
Avg loss: 0.14822134375572205 +/- 0.028701960327670934
Avg accuracy: 0.9453552007675171 +/- 0.01728020278573327
Avg sensitivity: 0.9422680412371133 +/- 0.0231442724955089
Avg specificity: 0.9488372093023255 +/- 0.02027394857460779
Avg f1-score: 0.9480723195968338 +/- 0.01658213359715215
```

Figura 5.93.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

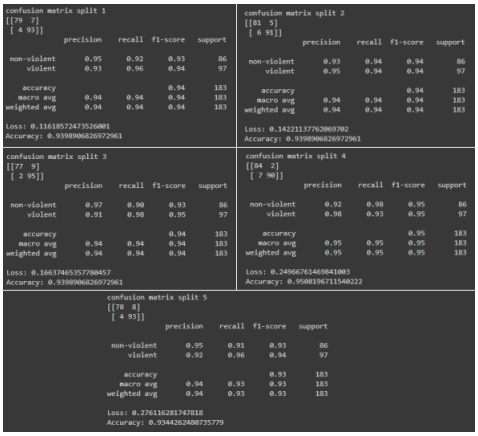

Figura 5.94.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer Adam e 50 epoche

Di seguito riportiamo la curva AUC prodotta sempre tramite 5-fold cross validation.

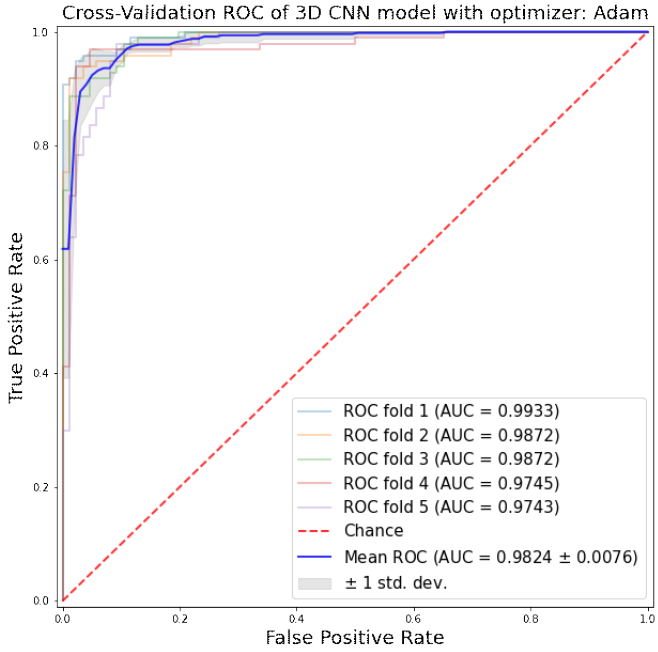

Figura 5.95.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer Adam e 50 epoche

A differenza del test effettuato con un numero minore di epoche, possiamo notare un piccolo decremento nell'accuracy.

```
Avg loss: 0.19011113047599792 +/- 0.062075169160129035
Avg accuracy: 0.9409835934638977 +/- 0.005354074424685833
Avg sensitivity: 0.9525773195876288 +/- 0.017974841004291423
Avg specificity: 0.9279069767441861 +/- 0.028859706153467096
Avg f1-score: 0.9448432283022121 +/- 0.004255330189782747
```

Figura 5.96.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine possiamo vedere le 5 matrici di confusione relative al test con AdaDelta.

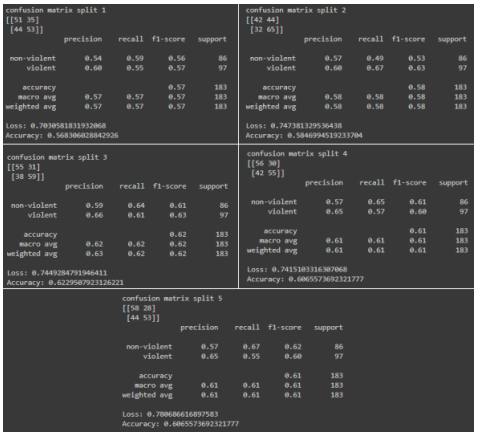

Figura 5.97.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 20 epoche

Anche su questo dataset, notiamo un incremento degli errori di tipo 1 e tipo 2. Riportiamodi seguito la curva AUC.

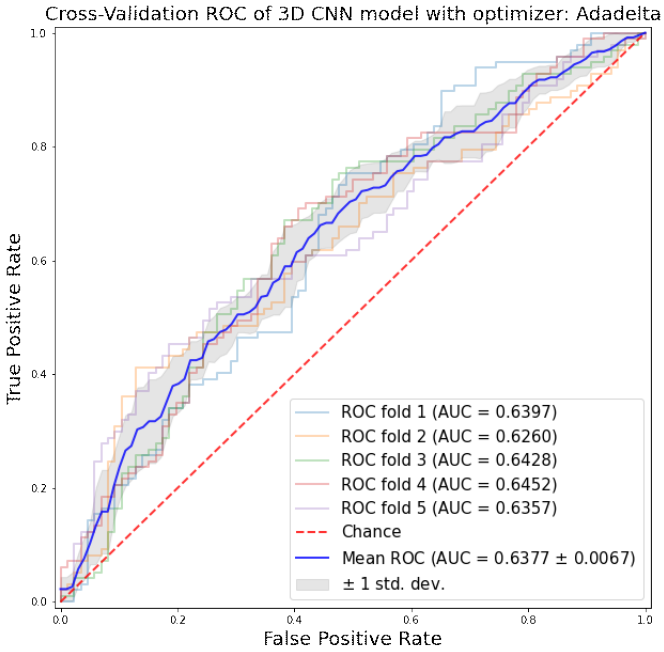

Figura 5.98.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 20 epoche

I risultati ottenuti, come mostrati nell'immagine sottostante, non sono ottimi, l'accuratezza media è uguale a 59,7%.

```
Avg loss: 0.7435129880905151 +/- 0.024655418193448134
Avg accuracy: 0.5978142023086548 +/- 0.019117861133947887
Avg sensitivity: 0.5876288659793814 +/- 0.047017543303057234
Avg specificity: 0.6093023255813954 +/- 0.06602357937943439
Avg f1-score: 0.6069916398583804 +/- 0.022148494210080926
```

Figura 5.99.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Riportiamo di seguito le 5 matrici di confusione, per il test effettuato con numero di epoche uguale a 50.

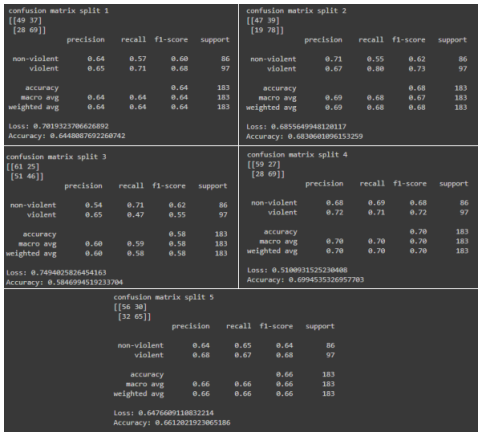

Figura 5.100.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 50 epoche

Anche qui notiamo un elevato numero di errori di tipo 1 e tipo 2. Nell'immagine sottostante è raffigurata la curva AUC.

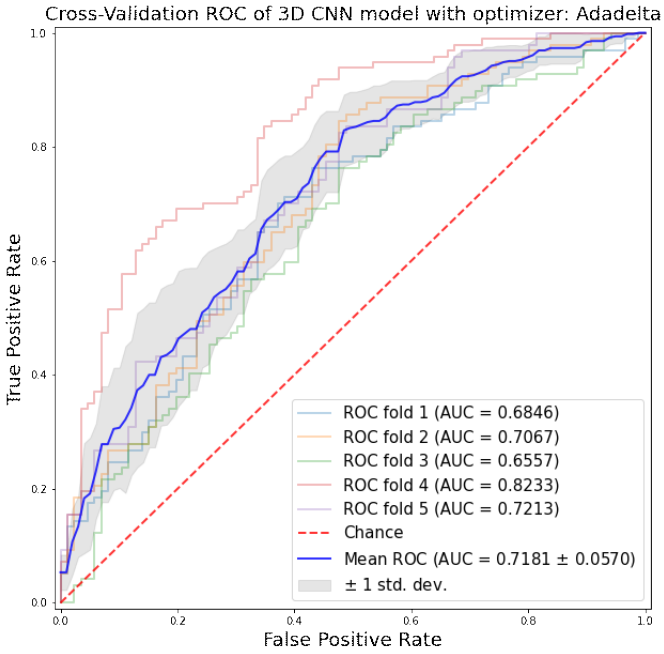

Figura 5.101.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 50 epoche

Possiamo notare un aumento rispetto al test con 20 epoche, infatti, l'accuratezza media è pari all' 65,4%. I risultati, comunque, sono molto bassi rispetto agli altri due optimizer.

```
Avg loss: 0.7435129880905151 +/- 0.024655418193448134
Avg accuracy: 0.5978142023086548 +/- 0.019117861133947887
Avg sensitivity: 0.5876288659793814 +/- 0.047017543303057234
Avg specificity: 0.6093023255813954 +/- 0.06602357937943439
Avg f1-score: 0.6069916398583804 +/- 0.022148494210080926
```

Figura 5.102.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split.

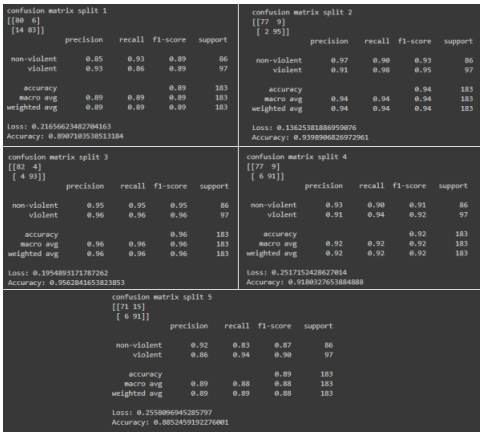

Figura 5.103.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer RMSProp e 20 epoche

Di seguito riportiamo la curva AUC prodotta tramite 5-fold cross validation.

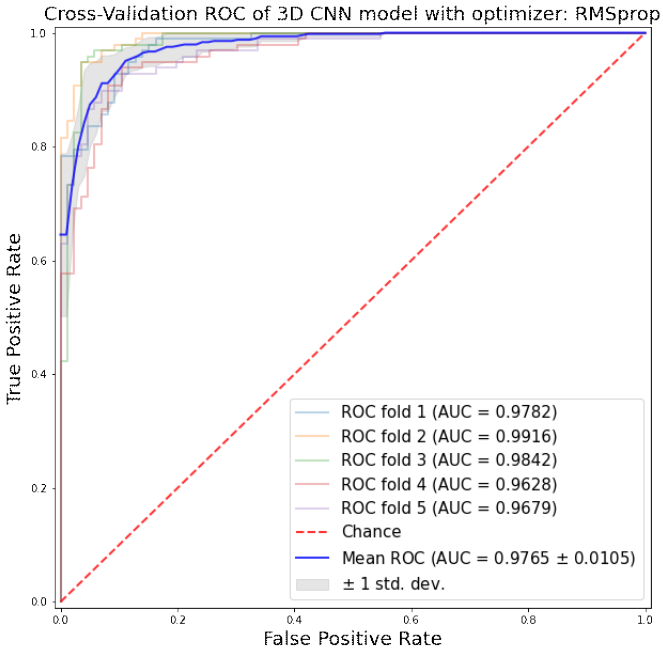

Figura 5.104.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer RMSProp e 20 epoche

Anche con questo optimizier, come possiamo notare dall'immagine, l'accuratezza media è del 91,8%.

```
Avg loss: 0.21116686165332793 +/- 0.04364414360797679
Avg accuracy: 0.9180327773094177 +/- 0.027431476824827706
Avg sensitivity: 0.934020618556701 +/- 0.0420537691842704
Avg specificity: 0.9 +/- 0.04325831450636807
Avg f1-score: 0.9233838457753081 +/- 0.026101723262994293
```

Figura 5.105.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer RMSProp e 20 epoche

Optimizer RMSProp 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

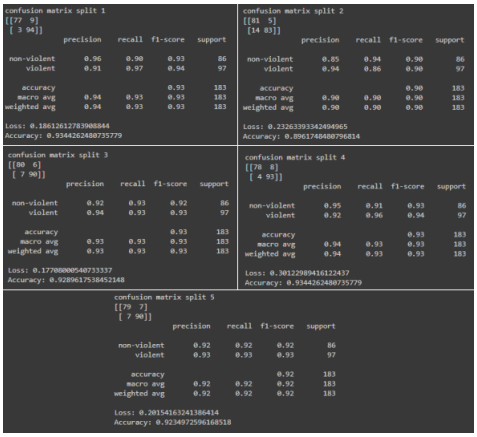

Figura 5.106.: Matrici di confusione sul Surveillance Camera Fight Dataset con optimizer RMSProp e 50 epoche

Nell'immagine sottostante è raffigurata la curva AUC.

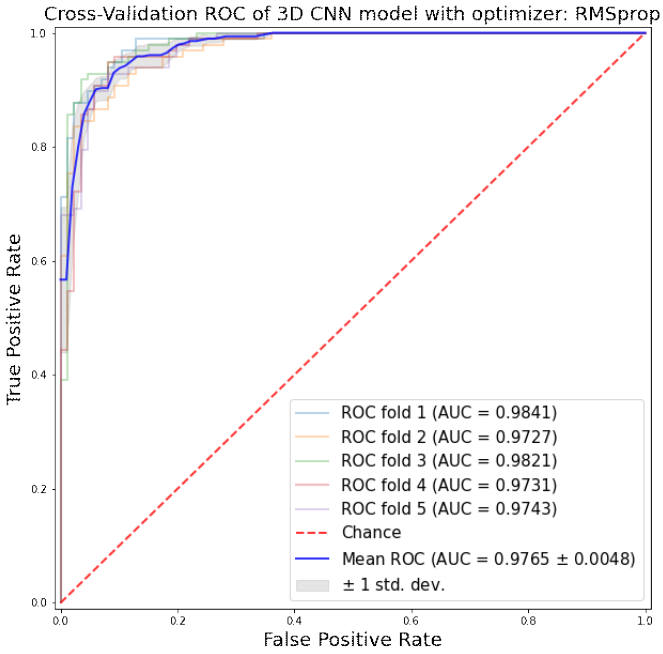

Figura 5.107.: Area Under Curve sul Surveillance Camera Fight Dataset con optimizer RMSProp e 50 epoche

Come si nota nell'immagine sottostante relativa ai risultati, l'accuratezza media è poco maggiore al caso con il numero di epoche pari a 20.

```
Avg loss: 0.21972231864929198 +/- 0.044917263784697174
Avg accuracy: 0.9234972715377807 +/- 0.014249634709367878
Avg sensitivity: 0.9278350515463918 +/- 0.03966058569416774
Avg specificity: 0.9186046511627908 +/- 0.016444343748524357
Avg f1-score: 0.9274337550568521 +/- 0.015725485909539334
```

Figura 5.108.: Risultati relativi al test sul Surveillance Camera Fight Dataset con optimizer RMSProp e 50 epoche

5.2.6. Test Real Life Violence Dataset

Per via delle limitazioni di Google Colab, spazio a disposizione e tempo limitato per l'utilizzo della GPU, si è dovuto, necessariamente, ridimensionare il dataset. Il numero di clip all'interno del dataset è stato portato a 600, 300 video violenti e 300 video di comportamenti normali. I video sono stati eliminati partendo dai video con il peso maggiore, così da ridurre anche lo spazio utilizzato per la creazione delle memmamap.

Optimizer Adam 20 Epoche

Di seguito sono riportate le 5 matrici di confusione relative al test effettuato con l'optimizer Adam e il numero di epoche pari a 20.

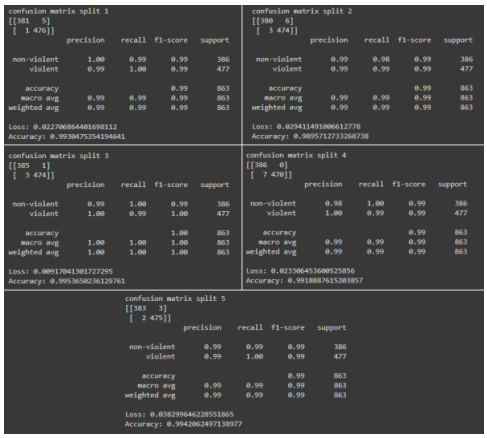

Figura 5.109.: Matrici di confusione sul Real Life Violence Dataset con optimizer Adam e 20 epoche

Nell'immagine sottostante viene riportata la curva AUC.

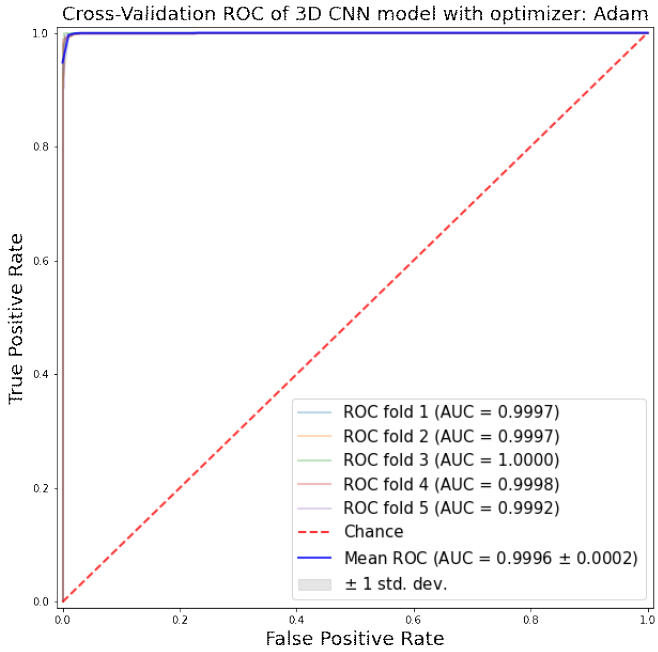

Figura 5.110.: Area Under Curve sul Real Life Violence Dataset con optimizer Adam e 20 epoche

Come si nota nell'ultima immagine relativa ai risultati ottenuti in questo test, il modello con l'optimizer Adam ha ottenuto un'accuratezza media pari a 99,2%.

```
Avg loss: 0.024578973650932312 +/- 0.009530560540661875
Avg accuracy: 0.9928157687187195 +/- 0.001993587039027272
Avg sensitivity: 0.9932914046121593 +/- 0.004275907348924776
Avg specificity: 0.9922279792746114 +/- 0.0059076446896328605
Avg f1-score: 0.9935007034076972 +/- 0.001798212081263839
```

Figura 5.111.: Risultati relativi al test sul Real Life Violence Dataset con optimizer Adam e 20 epoche

Optimizer Adam 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

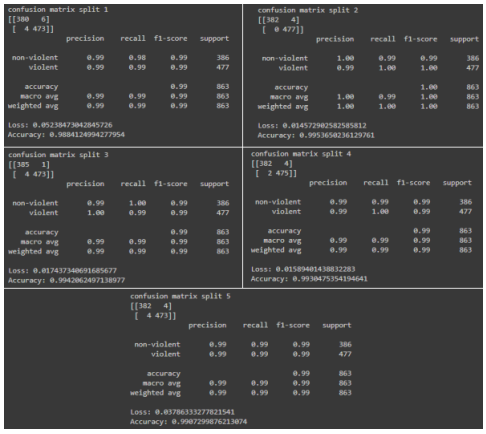

Figura 5.112.: Matrici di confusione sul Real Life Violence Dataset con optimizer Adam e 50 epoche

Di seguito riportiamo la curva AUC prodotta sempre tramite 5-fold cross validation.

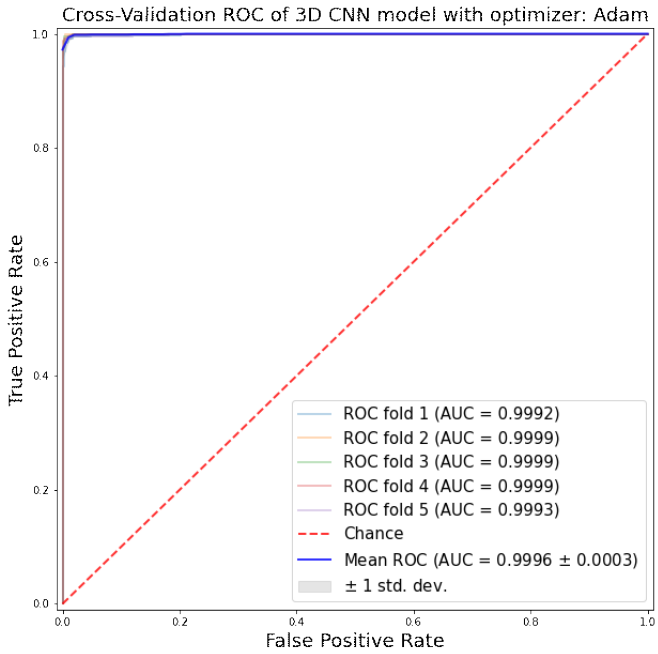

Figura 5.113.: Area Under Curve sul Real Life Violence Dataset con optimizer Adam e 50 epoche

I risultati ottenuti, sono uguali al test effettuato con un numero di epoche minori, infatti, come mostrato nell'immagine riportata di seguito, l'accuratezze media è uguale a 99,2%.

```
Avg loss: 0.027630464173853397 +/- 0.015030829865837104
Avg accuracy: 0.9923522591590881 +/- 0.0024960333087835776
Avg sensitivity: 0.9941299790356395 +/- 0.003354297693920355
Avg specificity: 0.9901554404145078 +/- 0.00414507772020726 9
Avg f1-score: 0.9930889730385877 +/- 0.0022535499953629376
```

Figura 5.114.: Risultati relativi al test sul Real Life Violence Dataset con optimizer Adam e 50 epoche

Optimizer AdaDelta 20 Epoche

Nell'immagine possiamo vedere le 5 matrici di confusione relative al test con AdaDelta.

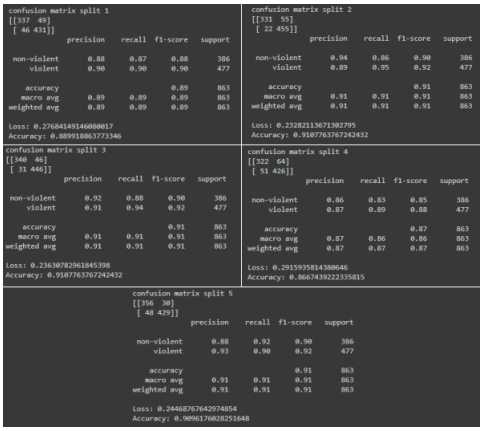

Figura 5.115.: Matrici di confusione sul Real Life Violence Dataset con optimizer AdaDelta e 20 epoche

Anche su questo dataset, notiamo un incremento degli errori di tipo 1 e tipo 2. Riportiamodi seguito la curva AUC.

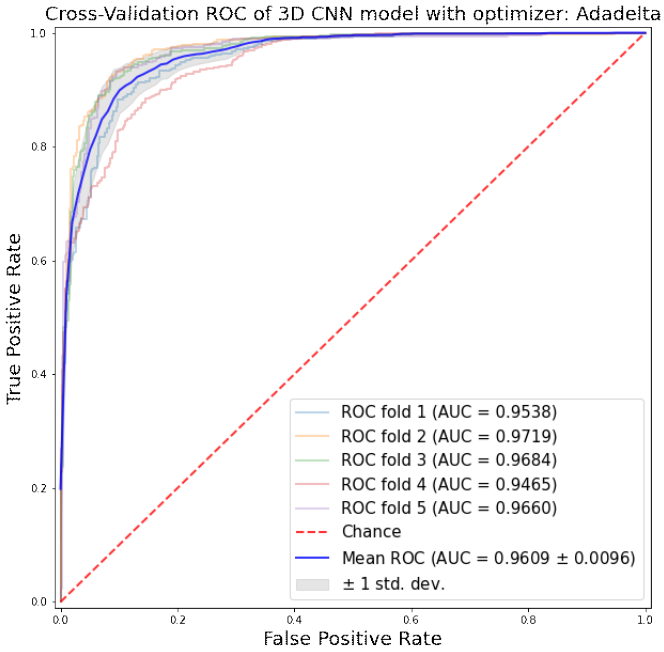

Figura 5.116.: Area Under Curve sul Real Life Violence Dataset con optimizer AdaDelta e 20 epoche

I risultati ottenuti, come mostrati nell'immagine sottostante, non sono ottimi, l'accuratezza media è uguale a 89,7

```
Avg loss: 0.25645034313201903 +/- 0.023466003934994997
Avg accuracy: 0.8975666284561157 +/- 0.017336368739627902
Avg sensitivity: 0.9169811320754716 +/- 0.023442617607537696
Avg specificity: 0.8735751295336787 +/- 0.029098696543210293
Avg f1-score: 0.9081992123283872 +/- 0.015527033929689677
```

Figura 5.117.: Risultati relativi al test sul Real Life Violence Dataset con optimizer AdaDelta e 20 epoche

Optimizer AdaDelta 50 Epoche

Riportiamo di seguito le 5 matrici di confusione, per il test effettuato con numero di epoche uguale a 50.

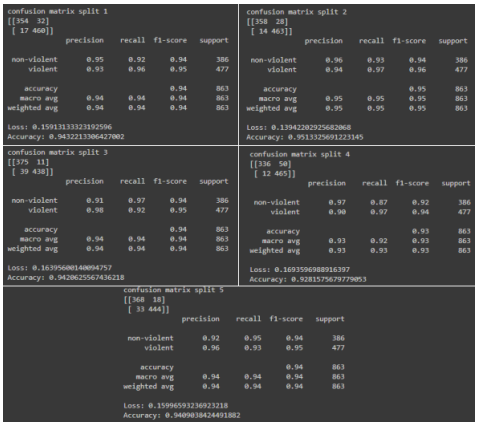

Figura 5.118.: Matrici di confusione sul Real Life Violence Dataset con optimizer AdaDelta e 50 epoche

Anche qui notiamo un elevato numero di errori di tipo 1 e tipo 2. Nell'immagine sottostante è raffigurata la curva AUC.

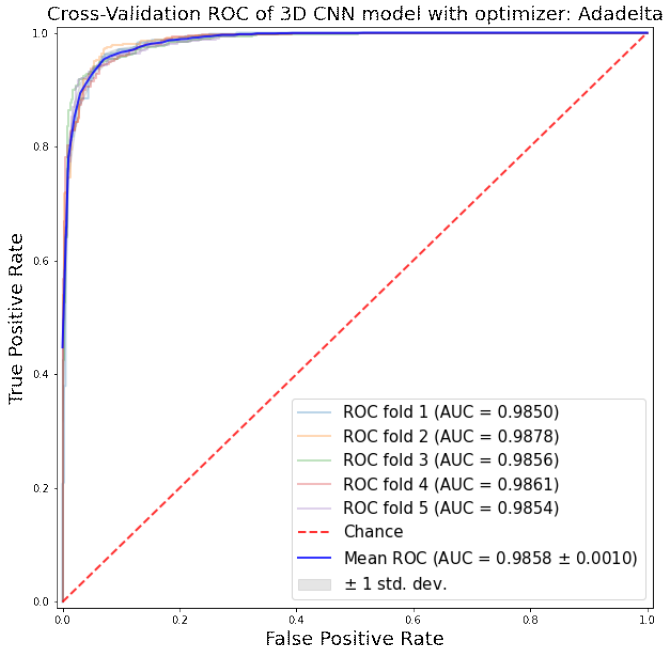

Figura 5.119.: Area Under Curve sul Real Life Violence Dataset con optimizer AdaDelta e 50 epoche

Possiamo notare un aumento rispetto al test con 20 epoche, infatti, l'accuratezza media è pari all' 94,1%. I risultati, comunque, sono molto bassi rispetto agli altri due optimizer.

```
Avg loss: 0.15836699903011323 +/- 0.010140306425870682
Avg accuracy: 0.941135573387146 +/- 0.007452123551380668
Avg sensitivity: 0.9517819706498951 +/- 0.022850191351679185
Avg specificity: 0.9279792746113988 +/- 0.034540587176163644
Avg f1-score: 0.9470470390803183 +/- 0.006184713729008675
```

Figura 5.120.: Risultati relativi al test sul Real Life Violence Dataset con optimizer AdaDelta e 50 epoche

Optimizer RMSProp 20 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split.

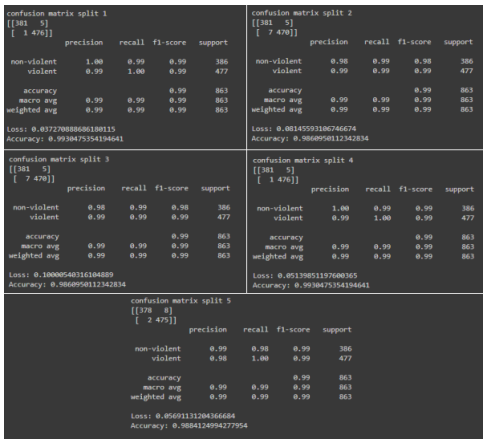

Figura 5.121.: Matrici di confusione sul Real Life Violence Dataset con optimizer RMSProp e 20 epoche

Come per il test effettuato con l'optimizer Adam, ci sono pochi errori di tipo 1 e di tipo 2. Di seguito riportiamo la curva AUC.

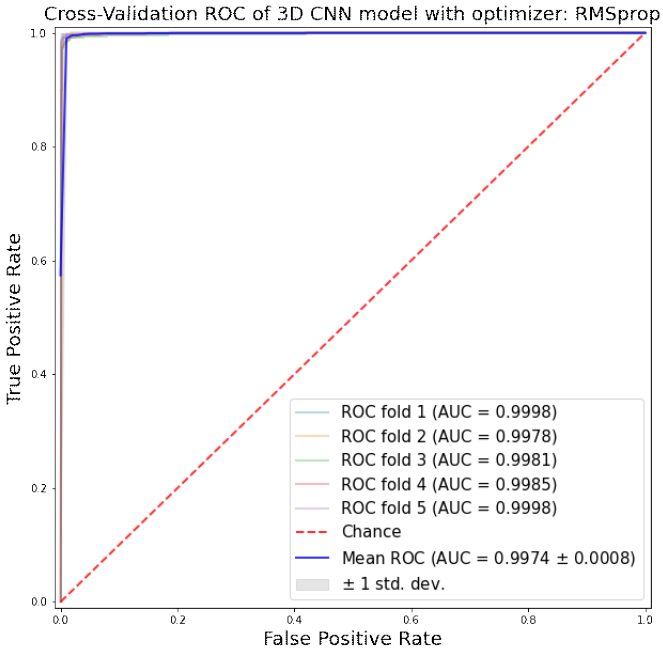

Figura 5.122.: Area Under Curve sul Real Life Violence Dataset con optimizer RMSProp e 20 epoche

Anche con questo optimizier, come possiamo notare dall'immagine, l'accuratezza media è del 98,9%.

```
Avg loss: 0.06540840938687324 +/- 0.02242620293408769
Avg accuracy: 0.9893395185470581 +/- 0.003143622054410176
Avg sensitivity: 0.9924528301886791 +/- 0.005870020964360594
Avg specificity: 0.9854922279792746 +/- 0.0031088082901554073
Avg f1-score: 0.9903694306532633 +/- 0.002863331072121212 4
```

Figura 5.123.: Risultati relativi al test sul Real Life Violence Dataset con optimizer RMSProp e 20 epoche

Optimizer RMSProp 50 Epoche

Nell'immagine sottostante sono riportate le 5 matrici di confusione relative ad ogni split, con il numero di epoche impostato a 50.

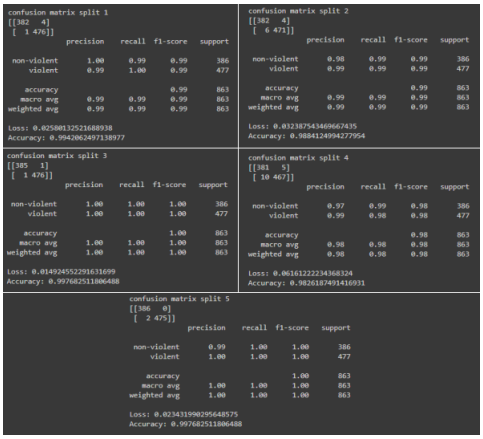

Figura 5.124.: Matrici di confusione sul Real Life Violence Dataset con optimizer RMSProp e 50 epoche

Nell'immagine sottostante è raffigurata la curva AUC.

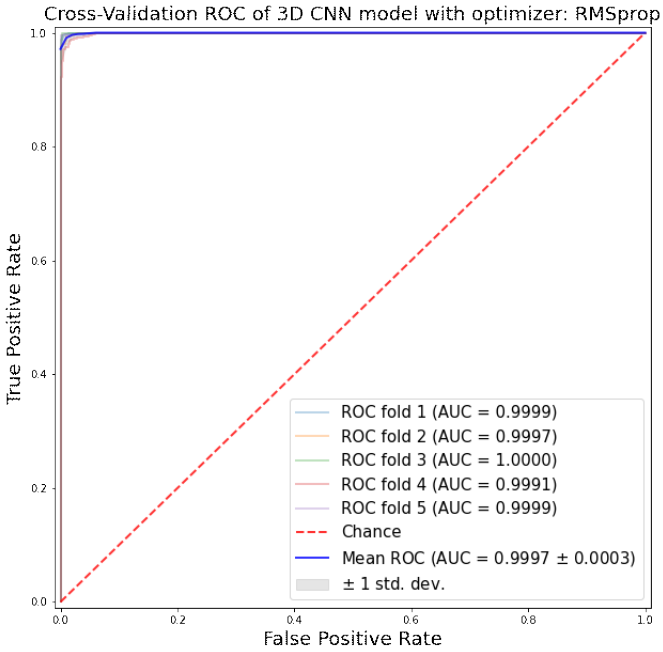

Figura 5.125.: Area Under Curve sul Real Life Violence Dataset con optimizer RMSProp e 50 epoche

Come si nota nell'immagine sottostante relativa ai risultati, l'accuratezza media è poco maggiore al caso con il numero di epoche pari a 20.

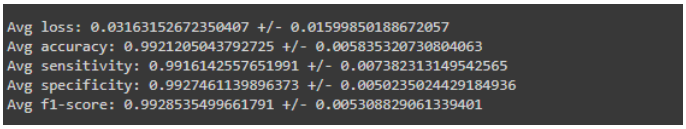

Figura 5.126.: Risultati relativi al test sul Real Life Violence Dataset con optimizer RMSProp e 50 epoche

5.3. Analisi dei risultati ottenuti

L'analisi dei dati mostra come i due metodi utilizzati ottengono alti valori di accuratezza nei vari dataset di benchmark testati.
Nell'architettura con classificatore a Support Vector Machine, si ricavano ottimi risultati, a conferma 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. Di seguito riassumiamo i valori migliori ottenuti per ogni dataset.

DATASET	**Parametri migliore**	**ACC ± SD**	**AUC**
Hockey Fight	'C'=1 'Kernel'=Linear	97,06 ± 0,82%	0,9965
Crowd Violence	'C'=10 'gamma'=10^{-4} 'Kernel'=rbf	99,01 ± 0,69%	1,0000
Dataset Intero	'C'=10 'gamma'=10^{-4} 'Kernel'=rbf	97,63 ± 0,51%	0,9980
AIRTLab Violence	'C'=100 'gamma'=10^{-4} 'Kernel'=rbf	96,07 ± 0,72%	0,9961
Surveillance Camera Fight	'C'=10 'gamma'=10^{-4} 'Kernel'=rbf	91,66 ± 2,18%	0,9844
Real Life Violence	'C'=100 'gamma'=10^{-4} 'Kernel'=rbf	99,29 ± 0,28%	0,9983

Tabella 5.1.: Tabella risultati architettura C3D + SVM

Inoltre, per i tre dataset testati anche da Accattoli, possiamo notare come i risultati ottenuti riguardo l'AUC sono molto simili. Questo è dovuto a diversi fattori:

- Ogni dataset è bilanciato;
- l'accuratezza alta tende ad assortigliarsi con l'AUC;

- il numero dei falsi positivi e falsi negativi risulta essere simili.

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, non solo che la rete ha un'accuratezza molto alta, ma che in determinate situazioni l'errore prodotto è dovuto ad input ambigui.

Per quanto riguarda i risultati ottenuti con l'architettura end-to-end, possiamo notare come la scelta degli iperparametri influenzi di molto i risultati. Infatti si nota una differenza abbastanza marcata tra l'optmizier AdaDelta e i restanti due optimizer Adam e RMSProp. Rispetto al numero di epoche, negli optmizer Adam e RMSProp, possiamo vedere come l'accuratezza media non cambia di molto tra i due test effettuati, questo è dovuto alla presenza della funzione EarlyStopping, di cui abbiamo discusso nel capitolo precedente. Riportiamo, nella pagina seguente, una tabella con la scelta migliore per ogni dataset e la relativa accuratezza ottenuta.

DATASET	**Optimizer migliore**	**ACC ± SD**	**AUC**
Hockey Fight	Adam, Epoche = 50	96,61 ± 0,62%	0,9934
Crowd Violence	RMSProp, Epoche = 20	99,28 ± 0,46%	0,9998
Dataset Intero	Adam, Epoche = 50	97,87 ± 0,56%	0,9974
AIRTLab Violence	Adam, Epoche = 50	95,62 ± 0,29%	0,9879
Surveillance Camera Fight	Adam, Epoche = 20	94,53 ± 1,72%	0,9853
Real Life Violence	Adam, Epoche = 20	99,28 ± 0,19%	0,9996

Tabella 5.2.: Tabella risultati architettura end-to-end

I casi peggiori si ottengono con l'optmizier AdaDelta, dovuto molto probabilmente al suo learning rate. Nei test effetuati con Adam ed RMSProp si ricavano ottimini risultati, in linea ad altre tecniche dello stato dell'arte.
Possiamo anche notare come le varie curve AUC siano molto simili a quelle ottenute dal test con architettura C3D e classificatore SVM.

In conclusione, possiamo affermare che i risultati ottenuti confermano pienamente 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 vari test su due architetture che utilizzano un modello di rete C3D.
La prima 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 affinché eseguisse la classificazione dei filmati in violenti o meno.
La seconda architettura è una combinazione di C3D come estrattore di funzionalità e due livelli completamente connessi per ottenere una rete end-to-end per la classificazione, dove solo i due layer finali fully connected vengono addestrati da zero sulle clip dei set di dati.

I test effettuati hanno evidenziato come una scelta degli Hyperparametrs influisca in maniera molto evidente sui risultati.
Purtroppo, come notato nel capitolo precedente, il sistema soffre nelle situazioni in cui all'interno della scena visiva sono inquadrati dei comportamenti amichevoli, ma che si presentano in maniera simile ad atti violenti. Questo potrebbe essere migliorato andando a prendere in considerazione l'intensità del movimento, combinando le feature estratte dalla rete con l'informazione riguardante l'accelerazione.

Inoltre, risulta anche difficile, se non impossibile, rappresentare la classe di "non aggressione". Mentre la violenza ha un comportamento ben specifico, il caso pacifico è rappresentato da un concetto troppo generico, che rende di fatto il sistema da preferire in

contesti ben specifici. Per poter migliorare anche questo aspetto si potrebbe fornire in input più esempi di comportamenti "normali", anche se questo può aumentare il numero di falsi negativi, che sono sicuramente degli errori molto peggiori. Un'altra soluzione può essere quella di prendere una decisione osservando la sequenza temporale dei rilevamenti prodotti dal sistema, pesando i casi positivi nel tempo.

In conclusione si può affermare che il sistema ha rispettato a pieno gli obiettivi proposti.

I lavori futuri che si delineano possono essere riassunti come segue:

- Introdurre informazioni aggiuntive (come l'accelerazione) per migliorare l'accuratezza e capacità discriminativa.
- Aggiungere diverse categorie di comportamenti violenti, implementando quindi un classificatore multi-classe.
- Testare nuovi optimizier presenti in letteratura.
- 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 a C3D per vedere se esistono reti in grado di ottenere performance ancora migliori dei sistema realizzati. Una possibile soluzione potrebbe essere una combinazione dell'architettura ConvLSTM e due livelli completamente connessi, ottenendo una rete end-to-end per la classificazione.

Appendice A.

OpenVino

OpenVino [40] è un toolkit sviluppato da Intel per facilitare l'inferenza più rapida dei modelli di deep learning. Aiuta gli sviluppatori a creare applicazioni di visione artificiale robuste e convenienti. Consente l'inferenza di deep learning all'edge e supporta l'esecuzione eterogenea su acceleratori di visione artificiale: CPU, GPU, Intel® Movidius™ Neural Compute Stick e FPGA. Supporta un gran numero di modelli di deep learning pronti all'uso.

In questo lavoro si è anche cercato di implementare una nuova architettura, sempre basata su una rete C3D, per la violence detection, prendendo spunto da un modello già implementato in [41].

Il modello prevedeva l'utilizzo di *OpenVino* per facilitare le operazioni di ottimizzazione e massimizzare le prestazioni.

Il processo di esecuzione di *Openvino* è il seguente:

- Forniamo un modello preaddestrato a *Model Optimizer*. Ottimizza il modello e lo converte nella sua rappresentazione intermedia (file .xml e .bin).
- Il motore di inferenza aiuta nella corretta esecuzione del modello su diversi dispositivi. Gestisce le librerie necessarie per eseguire correttamente il codice su piattaforme diverse.

I due componenti principali del toolkit *OpenVino* sono *Model Optimizer e Inference Engine*.

Il *Model Optimizer* è uno strumento da riga di comando multipiattaforma che facilita la transizione tra l'ambiente di formazione e quello di distribuzione. Regola i modelli di deep learning per un'esecuzione ottimale sui dispositivi di destinazione dell'endpoint.
Model Optimizer carica un modello in memoria, lo legge, costruisce la rappresentazione interna del modello, lo ottimizza e produce la rappresentazione intermedia. La rappresentazione intermedia è l'unico formato che il motore di inferenza accetta e comprende.

Dopo aver utilizzato il *Model Optimizer* per creare una rappresentazione intermedia (IR), utilizziamo il motore di inferenza per dedurre i dati di input. Il motore di inferenza è una libreria C++ con un set di classi C++ per dedurre i dati di input (immagini) e ottenere un risultato. La libreria C++ fornisce un'API per leggere la rappresentazione intermedia, impostare i formati di input e output ed eseguire il modello sui dispositivi. L'esecuzione eterogenea del modello è possibile grazie al motore di inferenza. Utilizza plug-in diversi per dispositivi diversi.

Il problema che abbiamo riscontrato è che con Google Colab, utilizzato nei nostri esperimenti, non potevamo creare finestre di dialogo esterne a quelle del notebook, per cui molte funzioni implementate nel toolkit non si potevano usare.

Bibliografia

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

[2] Zhihong Dong, Jie Qin, and Yunhong Wang. Multi-stream deep networks for person to person violence detection in videos. volume 662, pages 517–531, 11 2016.

[3] Chunhui Ding, Shouke Fan, Ming Zhu, Feng Weiguo, and Baozhi Jia. Violence detection in video by using 3d convolutional neural networks. volume 8888, pages 551–558, 12 2014.

[4] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:4489–4497, 2015.

[5] [Online] Vocabolario Treccani: `https://www.treccani.it/vocabolario/violenza`.

[6] S. Cappanera. *"Sviluppo e valutazione di un sistema di rilevamento automatico della violenza all ' interno di video"*, 2020. Tesi di Laurea Triennale, UNIVPM, A.A. 2019/20, Rel. Aldo Franco Dragoni.

[7] Ming Cheng, Kunjing Cai, and Ming Li. RWF-2000: An Open Large Scale Video Database for Violence Detection. pages 4183–4190, 2021.

[8] Seymanur Akti, Gozde Ayse Tataroglu, and Hazim Kemal Ekenel. Vision-based Fight Detection from Surveillance Cameras. *2019 9th International Conference on Image Processing Theory, Tools and Applications, IPTA 2019*, 2019.

[9] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[10] Ming Yang, Shuiwang Ji, Wei Xu, Jinjun Wang, Fengjun Lv, Kai Yu, Yihong Gong, Mert Dikmen, Dennis J. Lin, and Thomas S. Huang. Detecting human actions in surveillance videos. *2009 TREC Video Retrieval Evaluation Notebook Papers*, (October 2014), 2009.

[11] Graham W Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional Learning of Spatio-temporal Features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 140–153, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 495–502, Madison, WI, USA, 2010. Omnipress.

[13] Zhijun Fang, Fengchang Fei, Yuming Fang, Changhoon Lee, Naixue Xiong, Lei Shu, and Sheng Chen. Abnormal event detection in crowded scenes based on deep learning. *Multimedia Tools and Applications*, 75(22):14617–14639, 2016.

[14] Peipei Zhou, Qinghai Ding, Haibo Luo, and Xinglin Hou. Violent Interaction Detection in Video Based on Deep Learning. *Journal of Physics: Conference Series*, 844(1), 2017.

[15] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning Deep Representations of Appearance and Motion for Anomalous Event Detection. pages 8.1–8.12, 2015.

[16] Swathikiran Sudhakaran and Oswald Lanz. Learning to detect violent videos using convolutional long short-term memory. *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2017*, 2017.

[17] Zihan Meng, Jiabin Yuan, and Zhen Li. Trajectory-pooled deep convolutional networks for violence detection in videos. pages 437–447, 10 2017.

[18] Betsy George and Sangho Kim. *Spatio-temporal Networks: Modeling and Algorithms*. 01 2013.

[19] Tomaš Mikolov, Martin Karafiát, Lukaš Burget, Cernocky Jan, and Sanjeev Khudanpur. Recurrent neural network based language model. *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, (May 2014):1045–1048, 2010.

[20] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[21] Yoshua Bengio. *Learning deep architectures for AI*, volume 2. 2009.

[22] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. (November), 2012.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[24] Joe Yue Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:4694–4702, 2015.

[25] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Shuffle and learn: Unsupervised learning using temporal order verification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:527–544, 2016.

[26] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in Neural Information Processing Systems*, 1(January):568–576, 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] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, pages 675–678, 2014.

[33] Enrique Bermejo Nievas, Oscar Deniz Suarez, Gloria Bueno García, and Rahul Sukthankar. Violence detection in video using computer vision techniques. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6855 LNCS(PART 2):332–339, 2011.

[34] Tal Hassner, Yossi Itcher, and Orit Kliper-Gross. Violent flows: Real-time detection of violent crowd behavior. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 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.

[37] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[38] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. 2012.

[39] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[40] [Online] Available: https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html.

[41] Fath U.Min Ullah, Amin Ullah, Khan Muhammad, Ijaz Ul Haq, and Sung Wook Baik. Violence detection using spatiotemporal features with 3D convolutional neural network. *Sensors (Switzerland)*, 19(11):1–15, 2019.