UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Valutazione e ottimizzazione di reti neurali profonde per il riconoscimento di scene di violenza nei video basate su transfer learning

Evaluation and optimization of deep learning neural networks for violence detection in videos based on transfer learning

Candidato:
PAOLINI DAVIDE

Relatore:
Prof. DRAGONI ALDO FRANCO

Correlatore:
Dott. SERNANI PAOLO

Anno Accademico 2020-2021

Ringraziamenti

Un grazie speciale a Francesco, assieme al quale ho affrontato questi tre anni del mio percorso universitario, per tutte le volte che si è dimostrato disponibile ad aiutarmi e per tutti i momenti di divertimento passati insieme in facoltà.

Un grazie alla mia famiglia e ai miei amici per avermi sempre sostenuto e fatto compagnia nel momento del bisogno.

Grazie al professore Aldo Franco Dragoni per avermi dato la possibilità di svolgere la mia tesi su un tema così interessante, importante ed attuale.

Un ringraziamento speciale al mio correlatore Paolo Sernani per tutti i consigli e l'enorme aiuto offerto durante il periodo di tirocinio e scrittura della tesi.

Ancona, Ottobre 2021

PAOLINI DAVIDE

Sommario

Il crescente fenomeno di diffusione della violenza nella nostra società rende sempre più importante il problema della “violence detection”. Al momento le telecamere di sorveglianza sono lo strumento principalmente usato al fine di contrastare tale fenomeno, ma il loro monitoraggio è affidato ad operatori umani, i quali potrebbero non essere presenti o attenti nel momento in cui viene compiuto l’atto violento, inoltre non è possibile per una persona visionare l’enorme quantità di filmati che vengono prodotti quotidianamente. Per questo motivo si stanno sviluppando dei sistemi di intelligenza artificiale in grado di rilevare automaticamente la presenza di scene violente nei filmati. In questo lavoro di tesi vengono valutati diversi modelli di deep learning ottenuti con la tecnica del transfer learning al fine di verificare quale livello di prestazioni è possibile raggiungere nel campo della violence detection sfruttando tale tecnica e delle reti sviluppate e addestrate per risolvere problemi di diversa tipologia.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Negli ultimi anni si sta verificando un aumento di atti di violenza in luogo pubblico e di attacchi terroristici, ciò li rende un problema molto attuale e una grave minaccia alla sicurezza in tutto il mondo. Anche se oggi è possibile registrare questi eventi tramite telecamere di sorveglianza, o altri dispositivi come ad esempio gli smartphone, la quantità di video al secondo prodotta rende impossibile verificare manualmente la presenza di scene di violenza in essi, inoltre è raro che sia presente un operatore a monitorare una telecamera di sicurezza nel momento esatto in cui questa riprende un evento violento. Quindi con le esigenze di revisionare enormi quantità di video e di riuscire a rilevare in tempo reale lo svolgersi di un atto di violenza, per permettere l'intervento tempestivo di forze dell'ordine e soccorsi, è cresciuta l'importanza di sviluppare un sistema in grado di individuare automaticamente la presenza di scene violente nei video.

Nascono così gli studi riguardanti la *violence detection*, ovvero la rilevazione automatica ed efficace di scene di violenza all'interno di video di breve durata. La violence detection è una specializzazione della *human action recognition*, che è uno dei problemi più conosciuti nel campo della Computer Vision ed è stato ampiamente studiato. Infatti nella prima parte di questa tesi sarà usata una rete pre-addestrata sul dataset Sport1M in grado di determinare a quale sport appartengono le scene di un video. Questa rete era stata usata da S. Accattoli nel suo lavoro di tesi magistrale [1] per realizzare appunto un sistema per il riconoscimento automatico di aggressioni da parte di due o più persone con lo scopo di supportare i sistemi di sorveglianza nel rilevare crimini.

1.1 Obiettivi della tesi

Gli obiettivi di questo lavoro di tesi sono i seguenti:

- Testare l'accuratezza di modelli di classificazione di scene di violenza basati su transfer learning, confrontando diverse architetture di reti neurali profonde.
- Trovare la combinazione migliore degli iperparametri delle reti neurali profonde al fine di massimizzare l'accuratezza dei modelli confrontati.

1.2 Struttura della tesi

Il presente lavoro di tesi è suddiviso nel seguente modo:

- Nel secondo capitolo vengono illustrati gli approcci utilizzati nello stato dell'arte come soluzioni per il problema della violence detection, ponendo particolare attenzione sulle tecniche deep learning.
- Nel terzo capitolo viene esposta la teoria riguardante le reti convoluzionali, la C3D utilizzata, il transfer learning e vengono introdotti rapidamente tutti gli strumenti che sono stati utilizzati per realizzare gli esperimenti. Successivamente viene approfondita la struttura della rete utilizzata e ne viene mostrata l'implementazione; infine vengono riportati e analizzati i risultati degli esperimenti eseguiti.
- Nel quarto capitolo viene discussa la teoria delle reti LSTM, BiLSTM e ConvLSTM, inoltre vengono introdotte brevemente le Keras Applications usate. Successivamente vengono descritte le strutture delle varie reti utilizzate e mostrate le loro implementazioni, infine vengono riportati ed analizzati i risultati degli esperimenti.
- Nel quinto capitolo viene eseguita una analisi riassuntiva dei risultati ottenuti e vengono presentate le proposte per possibili sviluppi futuri.

Capitolo 2

Stato dell'Arte

La maggior parte degli approcci derivano da quelli usati per la human action recognition e utilizzano quindi tecniche di Computer Vision. Per l'identificazione di un atto violento è necessario estrarre dai video le informazioni spazio-temporali, ovvero l'insieme di informazione spaziale, informazione temporale e informazione del movimento, di cui ad esempio fa parte l'accelerazione che permette di distinguere un'aggressione da un contatto "amichevole", in genere caratterizzato da movimenti più lenti. Secondo Xu et al. [2] le tecniche possono essere divise in due classi a seconda delle features estratte dai video: nel caso di feature locali i video vengono rappresentati tramite l'uso di *point of interest*, mentre nel caso di feature globali vengono valutate per intero le caratteristiche di più frame. Ma negli ultimi anni sono state sviluppate nel campo dell'action recognition delle tecniche basate sul deep learning che hanno dimostrato in diversi studi di poter performare meglio di quelle usate in precedenza. In particolare Ji et al. in un articolo [3] propongono un modello composto da 3 layer convoluzionali e un layer fully-connected e addestrato sul dataset TRECVid2008, il quale consiste di video di sorveglianza registrati presso l'aeroporto di Londra-Gatwick (Figura 2.1). Mentre Yang et al. [4] hanno sviluppato un modello, addestrato sul dataset TRECVid2008, formato da un "head detector" basato su una rete convoluzionale 3D, un sistema di "head tracking", uno stadio di estrazione delle feature e una classificazione operata da due SVM e una 3D CNN (Figura 2.2). Infine Taylor et al. [5] hanno proposto un modello basato su convGRBM (Convolutional Gated Restricted Boltzmann Machine) addestrato sul dataset NORB, composto da immagini di giocattoli (Figure 2.3a e 2.3b).

Method	Boxing	Handclapping	Handwaving	Jogging	Running	Walking	Average
3D CNN	90	94	97	84	79	97	90.2
Schüldt [13]	97.9	59.7	73.6	60.4	54.9	83.8	71.7
Dollár [14]	93	77	85	57	85	90	81.2
Niebles [56]	98	86	93	53	88	82	83.3
Jhuang [16]	92	98	92	85	87	96	91.7
Schindler [53]	–	–	–	–	–	–	92.7

Figura 2.1: Confronto su dataset KTH tra il modello proposto da Ji et al. e alcuni modelli allo stato dell'arte

CellToEar	#Ref	#Sys	#CorDet	#FA	#Miss	Act.DCR	Min.DCR
NEC-1	194	35	3	32	191	**0.995**	**0.991**
NEC-2	194	20	1	19	193	1.001	0.998
NEC-3	194	20	1	19	193	1.001	0.998
UIUC-1	194	183	0	58	194	1.019	1.060
ObjectPut	#Ref	#Sys	#CorDet	#FA	#Miss	Act.DCR	Min.DCR
NEC-1	621	10	2	8	619	0.999	0.997
NEC-2	621	11	3	8	618	0.998	0.998
NEC-3	621	5	2	3	619	**0.998**	**0.997**
UIUC-1	621	555	1	190	620	1.061	1.020
Pointing	#Ref	#Sys	#CorDet	#FA	#Miss	Act.DCR	Min.DCR
NEC-1	1063	6	2	4	1061	0.999	0.999
NEC-2	1063	5	2	3	1061	**0.999**	**0.998**
NEC-3	1063	6	2	4	1061	0.999	0.999
UIUC-1	1063	774	13	225	1050	1.062	1.006

Figura 2.2: Performance ottenute dai metodi proposti da Yang et al.

Prior Art	Accuracy
HOG3D-KM-SVM	85.3
HOG/HOF-KM-SVM	86.1
HOG-KM-SVM	79.0
HOF-KM-SVM	88.0

Convolutional architectures	Accuracy
$32\text{convGRBM}^{16\times16}\text{-}128\text{F}^{9\times9\times9}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-log\_reg}$	88.9
$32\text{convGRBM}^{16\times16}\text{-}128\text{F}^{9\times9\times9}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-mlp}$	**90.0**
$32\text{F}^{16\times16\times2}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-}128\text{F}^{9\times9\times9}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-log\_reg}$	79.4
$32\text{F}^{16\times16\times2}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-}128\text{F}^{9\times9\times9}_{CSG}\text{-R/N/P}^{4\times4\times4}_{A}\text{-mlp}$	79.5

(a) Confronto su dataset KTH tra le architetture sviluppate da Taylor et al. e alcuni modelli allo stato dell'arte

Method	AP
Prior Art [27]:	
HOG3D+KM+SVM	45.3
HOG/HOF+KM+SVM	**47.4**
HOG+KM+SVM	39.4
HOF+KM+SVM	45.5
convGRBM+SC+SVM	**46.6**

(b) Confronto su dataset Hollywood2 tra le architetture sviluppate da Taylor et al. e alcuni modelli allo stato dell'arte

Figura 2.3

I successi ottenuti da queste tecniche hanno poi portato alla pubblicazione di diversi studi sul problema della violence detection in cui viene proposto l'utilizzo di reti neurali profonde. In uno studio Dong et al. [6] propongono l'uso di reti convoluzionali operanti su 3 distinti flussi di informazioni, le quali sono divisi in temporali, spaziali e relative all'accelerazione; similarmente Zhou et al. [7] hanno sviluppato una ConvNet chiamata FightNet, usando come base la rete TSN, la quale vanta eccellenti risultati nel campo dell'action recognition e decidendo di fornire in input i fotogrammi RGB, i campi *optical flow* e i campi di accelerazione (Figura 2.4a). Mentre Xu et al. [8] hanno realizzato un modello in cui vengono usati 3 stream, denominati "appearance", "motion" e "joint representation", il quale è stato ottenuto tramite *early fusion* dei primi due, ciascuno terminante in una SVM a singola classe, i cui output vengono infine combinati tra loro attraverso la tecnica di *late fusion*. Infine Fang et al. (Figura 2.4b) in un articolo [9] propongono un approccio dove vengono prima ottenute le informazioni spaziali e il *multi-scale histogram of optical flow* per poi estrarre le feature complesse tramite PCANet, una rete molto semplice usata nel campo dell'*image classification*, successivamente viene impiegata una SVM per eseguire la classificazione (Figura 2.4c).

In questi articoli troviamo degli esempi di approcci utilizzati per superare il problema dell'incapacità delle normali strutture convoluzionali di cogliere anche l'informazione temporale dai video, oltre a quella spaziale. In generale possiamo suddividere i modelli che sono stati sviluppati in due categorie:

- Spatio-Temporal Networks, a loro volta suddivisi in 3D Convolutional Neural Networks e Recurrent Neural Networks. Le prime, chiamate anche 3D ConvNet, lavorano su input tridimensionali, ovvero sequenze di più frame, attraverso filtri tridimensionali. Le RNN, invece, sono in grado di mantenere una memoria di stato grazie a delle connessioni a retroazione, dunque il comportamento della rete per un dato frame varia in base alle caratteristiche dei frame precedenti nella sequenza di input [10].

- Multiple Stream Networks, le quali sono composte da più reti parallele. Uno stream si caratterizza come un processo di apprendimento indipendente, dunque ogni stream è responsabile di una classificazione individuale dato un certo input. Infine le classificazioni sono combinate per definire una classificazione finale per il video. Il vantaggio è che con questa tecnica risultano ridotte le probabilità che le feature di alto livello dominino su quelle di livello più basso [11].

Algorithm	Ped1(frame)		Ped1(pixel)		Ped2	
	EER	AUC	EER	AUC	EER	AUC
MPPCA [□]	40%	59.0%	81%	20.5%	30%	69.3%
Social force [□]	31%	67.5%	79%	19.7%	42%	55.6%
Social force+MPPCA [□]	32%	66.8%	71%	21.3%	36%	61.3%
Sparse reconstruction [□]	19%	–	54%	45.3%	–	–
Mixture dynamic texture [□]	25%	81.8%	58%	44.1%	25%	82.9%
Local Statistical Aggregates [□]	16%	92.7%	–	–	–	–
Detection at 150 FPS [□]	15%	91.8%	43%	63.8%	–	–
Joint representation (early fusion)	22%	84.9%	47.1%	57.8%	24%	81.5%
Fusion of appearance and motion pipelines (late fusion)	18%	89.1%	43.6%	62.1%	19%	87.3%
AMDN (double fusion)	16%	92.1%	40.1%	67.2%	17%	90.8%

(a) Confronto su dataset UCSD tra il modello proposto da Xu et al. e alcuni modelli allo stato dell'arte

Features	Classifier	Dataset	
		Movies	Hockey
BoW (STIP) [7]	SVM	82.5%	88.6%
	AdaBoost	74.3%	86.5%
BoW (MoSIFT) [7]	SVM	84.2%	91.2%
	AdaBoost	86.5%	89.5%
Extreme Acceleration [17]	SVM	85.4%	90.1%
	AdaBoost	98.9%	90.1%
Motion Analysis [18]	Random Forest	98.5%	84.5%
FightNet	Softmax	**100%**	**97.0%**

(b) Confronto tra il modello proposto da Zhou et al. e alcuni modelli allo stato dell'arte

The training samples percentage	F_1 -measure									
	3×3PCAnet I + MHOF	3×3PCAnet SI	3×3PCAnet MHOF	5×5PCAnet SI + MHOF	5×5PCAnet SI	5×5PCAnet MHOF	SI + MHOF	SI	MHOF	OF
0.1	0.95841	0.96332	0.95733	0.97222	0.96644	0.96351	0.97666	0.97926	0.96586	0.96393
0.2	0.97710	0.97904	0.95766	0.98468	0.98134	0.98299	0.98604	0.98703	0.97773	0.97363
0.3	0.98767	0.99147	0.96415	0.99262	0.98966	0.98594	0.99010	0.99081	0.98118	0.97870
0.4	0.99613	0.99518	0.97482	0.99681	0.99495	0.99264	0.99533	0.99365	0.98819	0.98250
0.5	0.99699	0.99692	0.97905	0.99744	0.99624	0.99259	0.99703	0.99534	0.99004	0.98740
0.6	0.99757	0.99822	0.98459	0.99765	0.99803	0.99411	0.99736	0.99601	0.99108	0.99112
0.7	0.99738	0.99825	0.98564	0.99862	0.99812	0.99726	0.99825	0.99819	0.99237	0.99291
0.8	0.99868	0.99887	0.98680	0.99870	0.99794	0.99739	0.99850	0.99766	0.99475	0.99524
0.9	0.99629	0.99770	0.99224	0.99853	0.99926	0.99925	0.99831	0.99850	0.99512	0.99681

(c) Performance ottenute dal modello proposto da Fang et al.

Figura 2.4

2.1 Spatio-Temporal Networks

Ad esempio Ullah et alii [12] propongono l'utilizzo di una 3D CNN come estrattore di feature, sottolineando come una normale rete convoluzionale non sarebbe in grado di catturare l'informazione temporale, per cui è necessario usare un filtro tridimensionale su un cubo formato da una sequenza di fotogrammi. Infatti il dataset è stato suddiviso in sequenze di 16 frame di dimensioni 128x171, successivamente ridimensionati a 112x112, ottenendo così un input 3x16x112x112. Ispirandosi ad un modello sviluppato da Tran et al. [13] la rete è composta da 8 livelli convoluzionali, 5 di *max pooling*, 2 fully-connected e infine

un layer di output con funzione di attivazione Softmax. L'architettura è rappresentata nella Figura 2.5, dove sono anche riportati il numero di filtri, le dimensioni dei filtri e delle feature map per i layer convoluzionali e di pooling. La classificazione del video è compito dei layer fully-connected e del layer Softmax, il quale è formato da due soli neuroni in quanto il loro numero deve eguagliare il numero di classi, che in questo caso sono "video violento" e "video non violento".

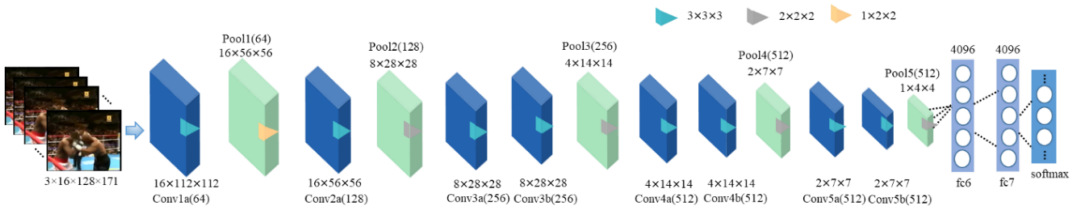


Figura 2.5: Rappresentazione grafica dell'architettura proposta da Ullah et al.

2.2 Multi-Stream Networks

Carneiro et al. [11] propongono un modello a quattro stream, i quali si occupano rispettivamente dei frame RGB, dell'*Optical Flow*, della *Depth Estimation* e del *Visual Rhythm*. In ciascuno stream viene usata una rete VGG-16, pre-addestrata sul dataset Imagenet per i primi 14 layer, con cui vengono calcolati i pesi da passare ai successivi 2 *dense layer*, che quindi ricevono le informazioni considerate importanti per distinguere un video violento da uno che invece non lo è. Infine per combinare le classificazioni dei singoli stream sono stati sperimentate tre diverse tecniche che includono l'uso di una SVM (Figura 2.6).

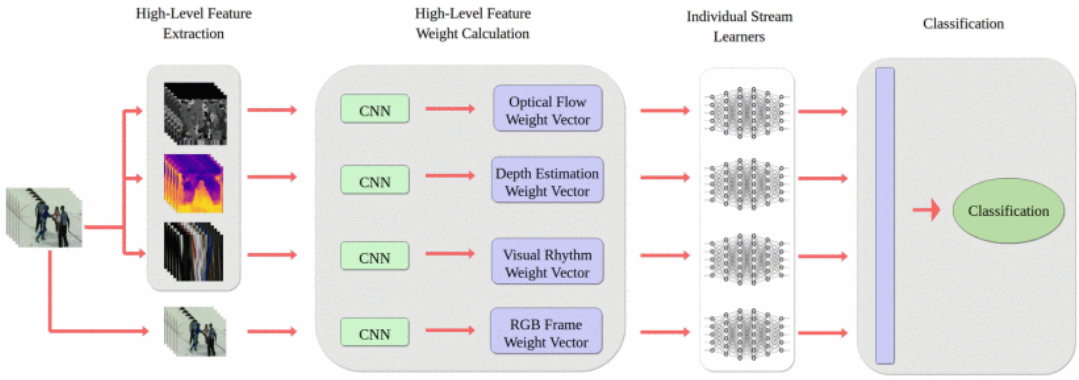


Figura 2.6: Rappresentazione grafica dell'architettura proposta da Carneiro et al.

2.3 Support Vector Machine

Le Support Vector Machine sono dei classificatori noti in letteratura per le loro ottime performance [14]. Le SVM sono dei modelli di machine learning supervisionato usati per la regressione e la classificazione tra due classi. Infatti dati un insieme di classi e un set di dati etichettati sono in grado di trovare una funzione di decisione e di usare tale funzione per classificare gli input non etichettati. Rispetto ad altre reti usate nella classificazione dimostra velocità più elevate e performance migliori per campioni di dati di piccole dimensioni. Per capirne meglio il funzionamento prendiamo un semplice esempio in cui sono presenti le due feature x, y e 2 classi: rosso e blu. Graficamente i dati si presentano come punti su un piano (Figura 2.7).

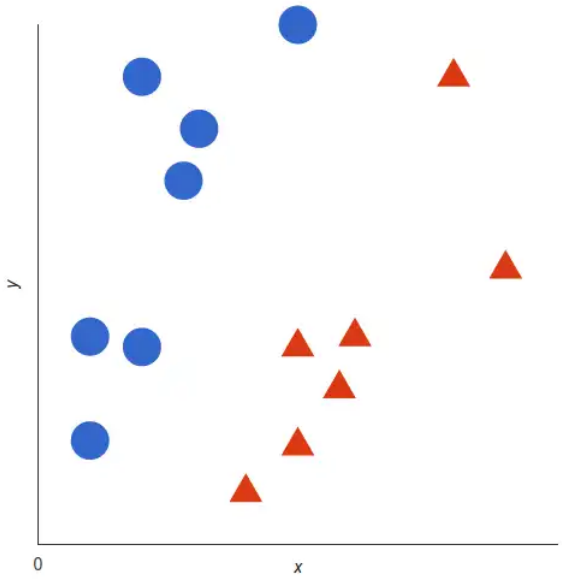


Figura 2.7: Rappresentazione grafica del dataset di esempio

La SVM trova l'iperpiano che meglio separa i punti blu da quelli rossi, in modo da poter poi classificare tutto ciò che si trova da una parte di tale iperpiano come punto blu e ciò che si trova dall'altra come punto rosso. L'iperpiano scelto è quello che massimizza la distanza tra l'elemento più vicino di ciascuna classe (Figura 2.8).

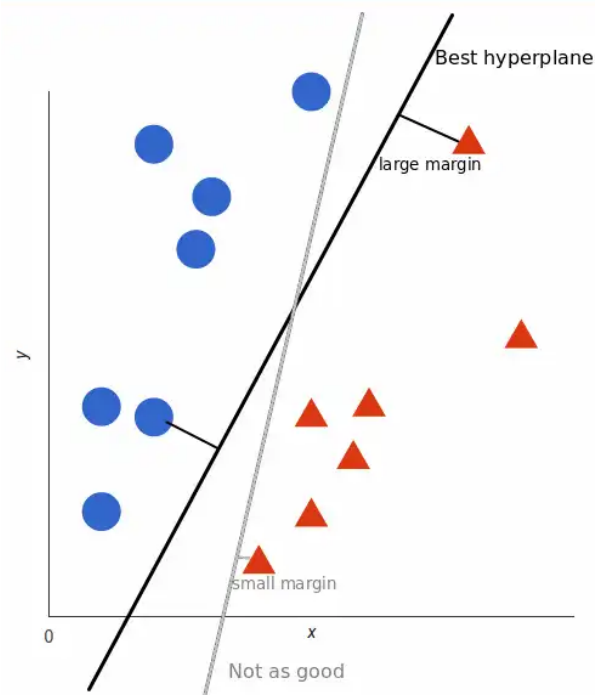


Figura 2.8: Esempio di miglior iperpiano di separazione

Capitolo 3

Esperimenti con Reti Neurali Feed-Forward

3.1 Strumenti usati

Nella prima parte di questo lavoro di tesi si è deciso di usare una rete convoluzionale 3D visti i vantaggi evidenziati nel capitolo precedente. In particolare è stata utilizzata la tecnica di transfer learning su un modello pre-addestrato chiamato C3D, il quale viene utilizzato per il problema della human action recognition.

3.1.1 Reti Convoluzionali (CNN)

Come riportato nel libro "Deep learning" [15] le reti *Multi Layer Perceptron* sono composte da un layer di input, uno di output e un certo numero di livelli intermedi detti *hidden layer*. Ognuno di questi strati è composto da un certo numero di neuroni e ciascun neurone è connesso con tutti i neuroni dello strato precedente, infatti questo tipo di connessione è detta *fully-connected* (Figura 3.1).

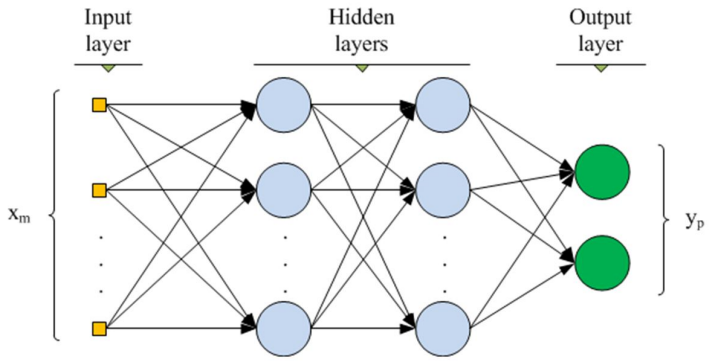


Figura 3.1: Esempio di rete Multi Layer Perceptron

Ad ogni connessione viene associato un peso (*weight*), utilizzato dal neurone che riceve l'informazione per processare quest'ultima. L'output prodotto da un neurone viene inviato

a tutti quelli dello strato successivo, fino a raggiungere l'output layer, il quale è incaricato di produrre l'output finale della rete. Il problema delle MLP è che il numero di connessioni può diventare molto elevato, ciò le rende praticamente inutilizzabili in diversi scenari, come ad esempio quelli in cui si lavora con immagini o video.

Le reti convoluzionali presentano invece un numero ridotto di connessioni, in quanto fanno uso di layer differenti dai fully-connected. A dare il nome a questo tipo di architettura è l'operazione matematica di convoluzione. Le CNN sono caratterizzate dalla connettività locale, per cui un neurone si connette ad un numero limitato di neuroni dello strato precedente. La regione di neuroni con cui si instaura una connessione dipende dal valore del campo ricettivo (*receptive field*), il quale è un parametro della rete. Inoltre nelle reti convoluzionali i pesi vengono condivisi tra più connessioni (*weight sharing*), rappresentano i parametri allenabili della rete e sono rappresentati mediante una matrice chiamata *kernel*. Queste due caratteristiche permettono di abbassare drasticamente il numero di connessioni presenti nella rete e il numero di pesi da mantenere in memoria rispetto ad una MLP. L'architettura di una CNN si basa sull'alternanza dei seguenti strati:

- Layer Convoluzionale
- Layer di Pooling
- Layer Fully-Connected
- Layer Softmax

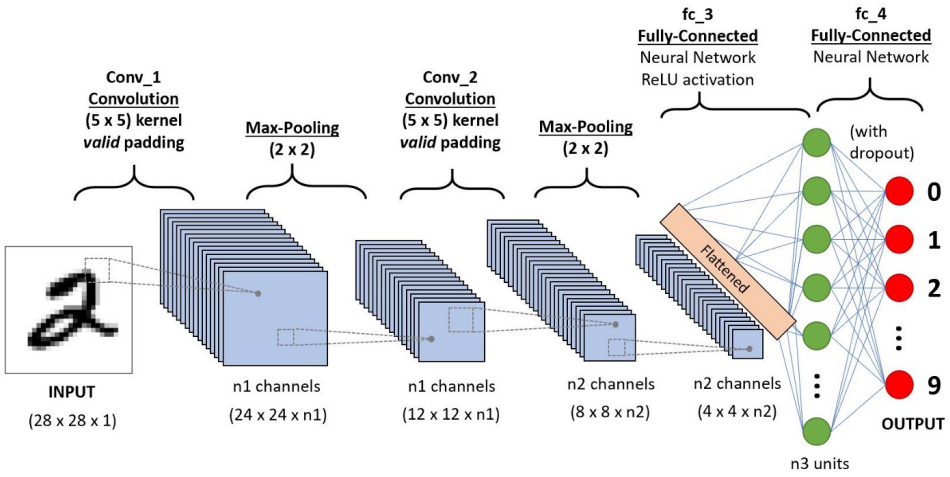


Figura 3.2: Esempio di rete convoluzionale utilizzata per riconoscere cifre scritte a mano

Layer Convoluzionale

È il layer che si occupa di estrarre le feature dalle informazioni in input. Il funzionamento di questo layer si basa sul far convolvere il kernel producendo in output una matrice che viene chiamata *feature map*. Il kernel non viene applicato sull'intero input, ma solo su una regione le cui dimensioni sono definite dal valore del receptive field (connettività locale), inoltre viene applicato lo stesso kernel su tutte le regioni possibili (weight sharing). La regione interessata dall'operazione di convoluzione viene fatta di volta in volta scorrere verso destra di un certo numero di pixel detto passo (*stride*) fino a raggiungere l'estremità destra dell'immagine, a questo punto viene riportata all'estremità sinistra e fatta scorrere verso il basso di un passo per poi reiterare il processo fino a coprire l'intera immagine. I filtri dei primi layer convoluzionali di una rete sono in grado di riconoscere schemi semplici come linee orizzontali, verticali e diagonali o curve, mentre negli ultimi strati vengono solitamente usati per estrarre feature complesse come la presenza di oggetti. I parametri da fissare per permettere il funzionamento del layer convoluzionale e determinare la dimensione della feature map in output sono:

- Depth: definisce il numero di filtri utilizzati, ed essendo che grazie ad ogni filtro è possibile estrarre una feature, rappresenta anche il numero di feature map.
- Stride: rappresenta il numero di elementi di cui il kernel scorre lungo l'input durante l'operazione di convoluzione. Maggiore è il valore di questo parametro minori saranno le dimensioni delle feature map.
- Zero-padding: consente di aggiungere bordi di 0 alle feature map, in quanto queste sono generalmente di dimensioni inferiori all'input su cui è stato applicato il filtro, quindi ha lo scopo di ripristinare le dimensioni della matrice.
- Kernel size: definisce le dimensioni della regione su cui vengono applicati i filtri.

Layer di Pooling

Il layer di pooling ha come scopo quello di ridurre progressivamente la dimensione delle feature map, mantenendo solo le informazioni ritenute utili. Esistono più operazioni di pooling, come ad esempio il Max Pooling e l'Average Pooling, che sono tra le più note. Come nel livello convoluzionale anche in questo layer viene definita una finestra, che scorre lungo la feature map e sugli elementi al suo interno viene applicata l'operazione di pooling. L'output di questo strato è una feature map, che continua a rappresentare quella ricevuta in ingresso pur essendo di dimensioni inferiori. I vantaggi derivanti dall'utilizzo del layer di pooling sono:

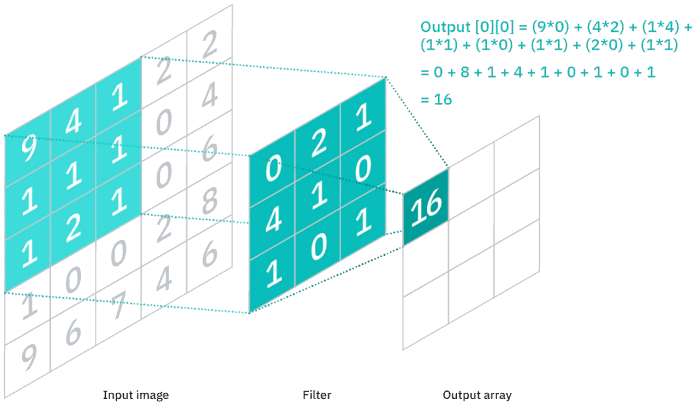


Figura 3.3: Esempio di operazione di convoluzione.

- Riducendo le dimensioni delle feature map si ottiene una diminuzione del numero di parametri e quindi del tempo di computazione.

- Diminuire il rischio di overfitting.

- Rendere le feature estratte invarianti rispetto alle trasformazioni affini come rotazioni, traslazioni e distorsioni. Infatti piccole trasformazioni nell'input non riescono a causare modifiche alla maggior parte degli output dell'operazione di pooling e quindi della nuova feature map.

Quest'ultima è una proprietà molto importante nei casi in cui si vuole semplicemente rilevare la presenza di una feature e non la posizione in cui si trova.

Layer Fully-Connected e Softmax

Questi ultimi due layer vengono usati per eseguire la classificazione. Lo strato fully-connected è una rete MLP, descritta in precedenza. Nell'ultimo livello FC della rete viene applicata la funzione di attivazione softmax, la quale restituisce per ogni classe la probabilità con cui l'input potrebbe appartenere a tale classe in modo tale che la somma delle probabilità sia uguale a 1. Prima di applicare la funzione di softmax vengono inseriti dei layer fully-connected, in quanto si è osservato che questi permettono di combinare le feature di alto livello provenienti dagli strati convoluzionali portando così a risultati migliori. L'unico parametro che è necessario specificare per questi layer è il numero di neuroni che li compongono. Notiamo che questi livelli lavorano sia in input sia in output con vettori monodimensionali quindi non possono essere seguiti da altri layer convoluzionali.

Training

L'addestramento di una rete convoluzionale è simile a quello delle reti neurali standard, ma presenta alcune modifiche necessarie a causa della presenza dei layer convoluzionali. Il processo di training può essere suddiviso nei seguenti passaggi:

1. Si inizializzano i parametri del kernel e i pesi in maniera casuale.
2. Si esegue la *forward propagation*: viene preso in input un elemento etichettato, questo viene processato e classificato dalla rete con pesi casuali, quindi produrrà probabilmente un risultato errato.
3. Si calcola l'errore totale tra l'output desiderato e quello ottenuto tramite una *loss function*, come ad esempio la Mean Squared Error.
4. Si calcola il gradiente dell'errore totale rispetto a tutti i pesi della rete propagandolo all'indietro attraverso i vari layer. Questo processo è chiamato *back propagation* perché ripercorre la rete all'indietro partendo dal livello di output.
5. Si aggiornano i pesi cercando di minimizzare il valore della loss function.

Il primo passaggio viene eseguito una sola volta all'inizio dell'addestramento, mentre i restanti passi vengono ripetuti per ciascun elemento fornito in input.

3.1.2 Reti Convoluzionali 3D

Le reti convoluzionali descritte finora sono in grado di apprendere solo l'informazione spaziale, infatti come affermato nello studio di Tran et al. [13] le CNN perdono l'informazione temporale dopo ciascuna operazione di convoluzione e pooling, anche nel caso in cui venga fornito in input una sequenza di frame, infatti tale sequenza verrebbe trattata dalla rete come un'immagine con più canali.

Convoluzione 3D e Pooling 3D

Per poter mantenere l'informazione temporale l'output delle operazioni di convoluzione e pooling deve essere tridimensionale e non più bidimensionale, a tale scopo è necessario usare dei filtri differenti. Infatti i filtri delle reti convoluzionali 3D non sono più rappresentati da matrici bensì da cubi e mantengono le proprietà di connettività locale e condivisione dei pesi. L'operazione di convoluzione 3D viene quindi eseguita su una sequenza di frame, ovvero convolvendo il kernel tridimensionale sul cubo formato da frame consecutivi impilati.

Anche nel caso dell'operazione di pooling 3D l'unica differenza sostanziale dalla versione bidimensionale è l'utilizzo di filtri tridimensionali, mentre le funzioni utilizzate sono le stesse.

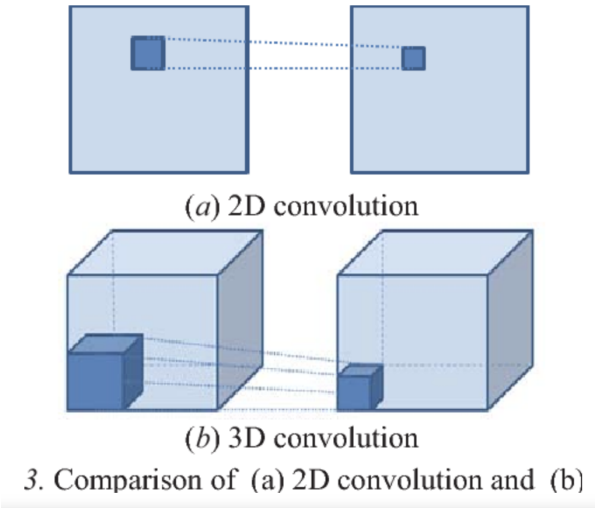


Figura 3.4: Esempio di operazione di convoluzione 3D.

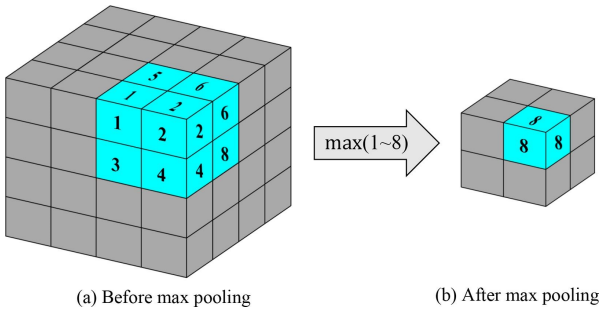


Figura 3.5: Esempio di operazione di max pooling 3D.

3.1.3 Transfer Learning

Sempre nel volume "Deep Learning" [15] il transfer learning viene descritto come un metodo, usato nel Machine Learning, che consiste nel riutilizzare un modello sviluppato per risolvere un determinato problema come punto di partenza per la realizzazione di un modello da utilizzare nella risoluzione di un problema differente, ma in cui ci interessa catturare caratteristiche simili a quelle estratte nel primo. Nei campi della Computer Vision e del Natural Language Processing viene molto utilizzato perché sono richieste grandi quantità di tempo e risorse computazionali per sviluppare reti in grado di raggiungere buone performance e i vari problemi condividono molte caratteristiche tra loro, ad esempio nel lavorare con delle immagini le reti devono sempre apprendere nozioni di basso livello come linee, semplici forme geometriche oppure gli effetti di trasformazioni geometriche o dell'illuminazione. Partire da un modello addestrato su un dataset di grandi dimensioni consente di concentrarsi sulle modifiche richieste per adattare tale modello al problema e di usare un dataset di dimensioni ridotte in fase di addestramento, risparmiando così tempo e risorse.

3.1.4 Modello C3D

C3D è un modello di rete pre-addestrato sviluppato dai ricercatori di Facebook per operare nel campo dell'action recognition [13]. Il modello è stato inizialmente addestrato su un dataset di dimensioni intermedie conosciuto in letteratura con il nome di UCF101 [16], in quanto usare dataset di dimensioni maggiori avrebbe richiesto tempi di addestramento molto più lunghi.

Architettura

Degli studi [17] dimostrano che usare kernel di dimensioni 3x3 nell'addestramento di reti neurali profonde porta alle performance migliori. Per questo motivo gli sviluppatori di C3D hanno deciso di usare un receptive field di dimensioni 3x3, eseguendo invece dei test per trovare il valore ottimale di profondità temporale dei kernel, in cui i risultati migliori sono stati ottenuti con kernel di dimensioni 3x3x3 e stride pari a 1x1x1. Dopo la fase di sperimentazione con il dataset UCF101, l'architettura ottenuta è composta da 8 layer di convoluzione, 5 layer max-pooling, 2 layer fully-connected ed infine un layer softmax. La rete lavora con input di dimensioni 3x16xHxW, dove 3 sono i canali Red, Green e Blue, 16 è la profondità temporale e infine H e W rappresentano altezza e larghezza dei fotogrammi del video.

Nella Figura 3.6 sono riportati i numeri di filtri utilizzati nei layer convoluzionali e il numero di neuroni dei layer fully-connected. Nei layer convoluzionali vengono usati progressivamente più filtri in quanto in questo modo si possono estrarre più feature di alto

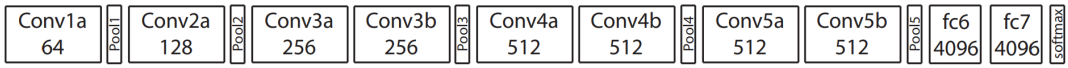


Figura 3.6: Rappresentazione dell'architettura finale del modello C3D.

livello a partire da poche feature di basso livello. Per quanto riguarda i layer di pooling vengono usati filtri di dimensione 2x2x2 con stride 2x2x2, ad eccezione del primo layer, in cui troviamo filtri di dimensione 1x2x2 con stride 1x2x2 al fine di preservare le informazioni temporali estratte. Per quanto riguarda il tipo di operazione di pooling è stato scelto il Max Pooling perché in alcuni studi [18] viene dimostrato come nel dominio del tempo sia da preferire agli altri tipi.

Addestramento

Per l'addestramento della rete C3D è stato usato il dataset Sport-1M, il quale è stato creato in collaborazione da Google e l'università di Stanford ed è composto da più di un milione di video di carattere sportivo presi da Youtube. L'utilizzo di un dataset di così grandi dimensioni permette alla rete neurale di avere una buona performance, inoltre il fatto che contenga dati legati allo sport, quindi diversi da quelli d'interesse per il problema della violence detection, favorisce la generalizzazione e aiuta a prevenire l'overfitting [19] [20]. Da ciascuno dei video presenti nel dataset sono state prelevate cinque clip della durata di 2 secondi in maniera casuale, queste clip sono state ridimensionate per ottenere frame 128x171 e ulteriormente ritagliate fino ad ottenere una dimensione 3x16x112x112. L'addestramento viene eseguito con l'algoritmo Stochastic Gradient Descent e con un *learning rate* variabile, che a partire dal valore 0,003 viene dimezzato ogni 150.000 iterazioni. La durata dell'addestramento è stato di 1.900.000 iterazioni, ovvero circa 13 epoche.

3.1.5 Tecnologie utilizzate

La realizzazione della rete e degli esperimenti si è svolta su Google Colaboratory [21] (abbreviabile in Colab), una piattaforma cloud di Google. Colab permette di scrivere codice Python da browser web per poi eseguirlo gratuitamente (o a pagamento se si necessita di più risorse) sui sistemi di Google con anche la possibilità di sfruttare le loro GPU e TPU per accelerare la computazione. La versione gratuita utilizzata in questo lavoro di tesi permetteva l'accesso a circa 12.7 GB di RAM e 68.40 GB di memoria secondaria. Per quanto riguarda la scrittura e l'esecuzione del codice Colab sfrutta i notebook Jupyter [22], ovvero dei documenti suddivisi in celle che possono contenere dei paragrafi testuali o delle righe di codice, e che consentono l'esecuzione separata e nell'ordine desiderato delle diverse celle di codice. L'uso dei notebook Jupyter è molto popolare nel campo dell'Intelligenza Artificiale e della Data Science in quanto permettono di definire dati,

scrivere funzioni, raccogliere risultati, compiere analisi, generare grafici e commentare passo per passo ciascuno di questi passaggi all'interno dello stesso documento. Il linguaggio di programmazione utilizzato in questo lavoro di tesi è appunto il Python, nella sua versione 3.6.9, che ha tra i diversi vantaggi la grande disponibilità di librerie pensate per Machine e Deep Learning. Tra queste sono state utilizzate Keras [23], OpenCV [24] e Sklearn [25]. Keras è una libreria open source ideata per sviluppare e valutare reti neurali profonde. Si basa su altre librerie come Tensorflow e Theano per ottenere un alto livello di astrazione e consentire una maggiore facilità d'uso all'utente, il quale può realizzare una rete in poche e semplici righe di codice. Nella fase di preprocessing è stata di fondamentale aiuto OpenCV, ovvero una libreria open source realizzata in C++ e utile nei campi della Computer Vision, del Machine Learning e dell'Image Processing. Scikit-learn, abbreviata in Sklearn, è una libreria open source che contiene numerose funzioni utili nel campo dell'apprendimento automatico, come ad esempio algoritmi di apprendimento classificatori o regressori oltre a diversi dataset. In questo lavoro è stata utilizzata per l'addestramento delle reti e per quantificare la qualità delle predizioni.

3.1.6 Dataset AirtLab

Il dataset per la violence detection realizzato dall'AIRTLab [26] è composto da 350 video etichettati in due classi: video violenti e video non violenti. Alcuni dei video etichettati come non violenti contengono però interazioni tra le persone nella scena, come ad esempio degli abbracci o pacche amichevoli, inserite per verificare la capacità della rete di gestire falsi positivi. Il dataset non è bilanciato infatti i video violenti sono 230, mentre i video non violenti sono 120. I video sono stati registrati da due diverse fotocamere e sono suddivisi, in modo bilanciato, nelle due cartelle cam1 e cam2. Tutte le clip sono in codifica MP4 con una risoluzione di 1920x1080 e frame rate di 30 fps. Tutte le scene sono state filmate tramite la fotocamera di un Asus Zenfone Selfie ZD551KL e una action camera TOPOP OD009B all'interno della stessa stanza con un'illuminazione naturale [27].

3.2 Realizzazione

Possiamo suddividere il sistema di riconoscimento utilizzato in tre elementi principali:

- Preprocessing dell'input
- Feature Extractor
- Classificatore

3.2.1 Input Preprocessing

Siccome la rete C3D accetta input di 16 frame è quindi necessario suddividere ogni video in più segmenti di 16 fotogrammi, ma ci sono alcuni problemi di cui tenere conto. Innanzitutto i video hanno durata variabile, inoltre nei video etichettati come violenti la violenza non è continua, perciò esistono delle sequenze che non contengono atti violenti, ma che comunque appartengono alla classe dei video violenti. La tecnica utilizzata è ripresa dalla tesi di Accattoli [1], quindi si cercano di utilizzare i filmati per intero per massimizzare la dimensione dell'input e non perdere informazioni utili. A tale scopo i filmati vengono divisi in modo sequenziale in segmenti di 16 frame che vengono etichettati allo stesso modo del video dai cui derivano. Essendo che i filmati generalmente non sono composti da multipli di 16 fotogrammi, i frame finali vengono scartati, ma ciò non rappresenta un problema in quanto è stato osservato che negli ultimi frame non sono contenute informazioni rilevanti. Infine le sequenze di frame e le relative etichette vengono salvate su due *memmap* per far sì che non vengono caricate tutte contemporaneamente in memoria.

3.2.2 Estrazione Feature

Per estrarre le feature si è deciso di utilizzare la rete pre-addestrata C3D, descritta nel capitolo 3.1.4. Come output della rete è stato preso il primo strato fully-connected, denominato "fc6", in quanto si tratta di un vettore monodimensionale di 4096 elementi che rappresenta la combinazione delle feature estratte da tutti i layer convoluzionali della C3D. Non è stato scelto il livello successivo in quanto l'output sarebbe stato ottenuto da ulteriori elaborazioni eseguite con lo scopo di classificare il video come una delle classi sportive definite nel dataset di addestramento della rete, portando ad un peggioramento delle performance perché le caratteristiche che contraddistinguono i diversi sport sono di tipo diverso da quelle che contraddistinguono un atto violento.

3.2.3 Classificazione

Nella fase di scelta del classificatore viene preso un approccio differente da quello proposto da Accattoli [1], dove viene implementata una Support Vector Machine. In questo lavoro di tesi le feature estratte vengono passate ad uno strato fully-connected di 512 neuroni, il cui output è poi passato ad un singolo neurone con la funzione di attivazione sigmoide, la quale si occupa di classificare i video in violenti o non violenti. Questa parte della rete sarà poi modificata nel corso degli esperimenti per provare a raggiungere performance migliori.

3.2.4 Implementazione

La seguente è quindi l'architettura del sistema di riconoscimento utilizzato negli esperimenti:

Layer Type	Output Shape	Parameter #
Conv3D, *3x3x3, stride=1*	(None, 16, 112, 112, 64)	5248
MaxPooling3D, *1x2x2*	(None, 16, 56, 56, 64)	0
Conv3D, *3x3x3, stride=1*	(None, 16, 56, 56, 128)	221312
MaxPooling3D, *2x2x2*	(None, 8, 28, 28, 128)	0
Conv3D, *3x3x3, stride=1*	(None, 8, 28, 28, 256)	884992
Conv3D, *3x3x3, stride=1*	(None, 8, 28, 28, 256)	1769728
MaxPooling3D, *2x2x2*	(None, 4, 14, 14, 256)	0
Conv3D, *3x3x3, stride=1*	(None, 4, 14, 14, 512)	3539456
Conv3D, *3x3x3, stride=1*	(None, 4, 14, 14, 512)	7078400
MaxPooling3D, *2x2x2*	(None, 2, 7, 7, 512)	0
Conv3D, *3x3x3, stride=1*	(None, 2, 7, 7, 512)	7078400
Conv3D, *3x3x3, stride=1*	(None, 2, 7, 7, 512)	7078400
ZeroPadding3D	(None, 2, 8, 8, 512)	0
MaxPooling3D, *2x2x2*	(None, 1, 4, 4, 512)	0
Flatten	(None, 8192)	0
Dense, *4096 units, ReLU activation*	(None, 4096)	33558528
Dropout, *0.5*	(None, 4096)	0
Dense, *512 units, ReLU activation*	(None, 512)	2097664
Dropout, *0.5*	(None, 512)	0
Dense, *1 unit, Sigmoid activation*	(None, 1)	513

Figura 3.7: Rappresentazione dell'architettura finale del sistema di riconoscimento.

Del preprocessing dell'input si occupa la funzione "preprocessVideos" che prende come parametri: il percorso alla base della repository di video (la quale deve contenere le due cartelle "violent" e "non-violent" a loro volta suddivise nelle cartelle "cam1" e "cam2") e il percorso in cui si desidera salvare i due array numpy contenenti i campioni dei filmati e i relativi label. Inizialmente viene richiamata la funzione "count_chunks" per poter creare delle memmap in grado di contenere esattamente tutti i campioni di 16 frame estraibili, successivamente si procede, con l'ausilio dei metodi messi a disposizione dalla libreria OpenCV, all'estrazione vera e propria delle sequenze di frame, le quali vengono infine memorizzate nella memmap "npSamples" e con la relativa etichetta salvata in "npLabels".

```
def count_chunks(videoBasePath):
    folders = ['violent', 'non-violent']
    cams = ['cam1', 'cam2']
    cnt = 0
```

```
    for folder in folders:
        for camName in cams:
            path = os.path.join(videoBasePath, folder, camName)

            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
                cnt += chunks
    return cnt
```

```
def preprocessVideos(videoBasePath, featureBasePath, verbose=True):
    folders = ['violent', 'non-violent']
    cams = ['cam1', 'cam2']
    total_chunks = count_chunks(videoBasePath)
    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 folder in folders:
        for camName in cams:
            path = os.path.join(videoBasePath, folder, camName)

            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 folder == 'violent':
                    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
```

L'implementazione del modello C3D è disponibile in due versioni, una che sfrutta le API funzionali di Keras mentre l'altra quelle sequenziali, in un repository github [28] in cui sono presenti anche i pesi ottenuti dall'addestramento sul dataset Sport-1M. In questo lavoro di tesi è stata utilizzata l'implementazione che utilizza le API sequenziali.

```
model = Sequential()
input_shape = (16, 112, 112, 3) #Input di 16 frame 112x112x3
```

```
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(name = 'flat5'))
# 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'))
```

Le feature estratte dalla C3D vengono fornite in input ad un layer fully-connected di 512 neuroni seguito da un altro layer con un singolo neurone che si occupa di eseguire la classificazione finale tramite la funzione di attivazione sigmoide.

```
def getC3DCNNModel(verbose=True):
    pretrainedModel = getFeatureExtractor('weights/weights.h5', 'fc6', False)
    for layer in pretrainedModel.layers:
        layer.trainable = False

    dropout1 = Dropout(.5)(pretrainedModel.output)
    fc7 = Dense(512, activation='relu', name='fc7')(dropout1)
    dropout2 = Dropout(.5)(fc7)
    output = Dense(1, activation='sigmoid')(dropout2)
    model = Model(inputs=pretrainedModel.inputs, outputs=output)
    if verbose:
        model.summary()
    opt = tf.keras.optimizers.Adam()    #Scelta Optimizer
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

    del pretrainedModel

    return model
```

Eseguire l'addestramento e i test sugli stessi dati porterebbe ad un risultato perfetto, in quanto alla rete verrebbe richiesto di predire i label dei campioni che ha appena visionato durante l'addestramento, mentre fallirebbe nel classificare correttamente campioni mai visti. Per evitare questo fenomeno, chiamato *overfitting*, viene utilizzata una procedura chiamata *cross-validation*, in cui una parte del dataset viene usata per la valutazione finale ed è chiamata *test set*, mentre la restante parte, secondo l'approccio *k-fold*, viene suddiviso in k *folds* e chiamato *training set*. Per k volte viene poi ripetuto il seguente procedimento: vengono usate k-1 parti del training set per l'addestramento e la rimanente parte viene usata per la validazione. Infine il modello viene valutato sul test set. In questo lavoro di tesi è stato utilizzato lo "Stratified ShuffleSplit cross-validator" della libreria SkLearn, in quanto include anche la strategia *ShuffleSplit* che permette di risolvere il problema del caricamento in memoria del dataset ordinato per classe, cosa che porterebbe a split estremamente sbilanciati. Per prevenire l'overfitting è stata impiegata anche la tecnica di *Early Stopping*, la quale permette di arrestare l'addestramento anticipatamente per evitare che i parametri della rete si adattino troppo ai dati di training, quindi consente di mantenere una maggiore capacità di generalizzazione.

```
def runEndToEndExperiment(getModel, batchSize, datasetBasePath, mmapDatasetBasePath,
     samplesMMapName, lablesMMapName, endToEndModelName, rState):
    chunk_number = count_chunks(datasetBasePath)
    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, random_state =
        rState)

    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(i==1)

        es = EarlyStopping(monitor='loss', mode='min', patience=5, verbose=1,
            restore_best_weights=True)
```

```
model.fit(X_train, y[train], validation_split=0.125, epochs=50, batch_size=
    batchSize, verbose=1, callbacks=[es])
```

3.3 Esperimenti

3.3.1 Metriche di valutazione

Per valutare le performance del sistema realizzato verrà data grande importanza ai parametri di accuratezza e *F1-score*, ma saranno considerati anche altri due importanti parametri: *sensitivity* e *specificity*. La sensitivity, o True Positive Rate, è definita come il rapporto tra il numero di True Positives e tutti gli elementi positivi: $TPR = \frac{TP}{TP+FN}$, mentre la specificity, o True Negative Rate, è definita come il rapporto tra il numero di True Negatives e tutti gli elementi negativi: $TNR = \frac{TN}{TN+FP}$. Nel nostro caso indicano rispettivamente la capacità del sistema di riconoscere come violento un video che effettivamente contiene una scena di violenza e la capacità del sistema di classificare correttamente come non violento un video in cui non compaiono atti violenti. Quindi in caso di bassi valori di sensitivity il sistema non ha riconosciuto molti video violenti come tali e in una situazione reale ciò porterebbe ad ignorare dei casi di aggressione; mentre in caso di bassi valori di specificity il sistema ha etichettato come violenti dei video che invece non lo sono e ciò in una situazione reale porterebbe a dei falsi allarmi. L'F1-score viene calcolato tramite la media armonica di sensitivity e *precision* ($PPV = \frac{TP}{TP+FP}$): $F1 = 2 * \frac{PPV*TPR}{PPV+TPR} = \frac{2*TP}{2*TP+FP+FN}$.

3.3.2 Ottimizzatori

Inizialmente sono state testate le prestazioni offerte da alcuni degli ottimizzatori disponibili nella libreria Keras. Gli algoritmi di ottimizzazione determinano in che modo aggiustare i parametri della rete al fine di minimizzare la loss function. Gli optimizer utilizzati sono SGD (*Stochastic Gradient Descent*), RMSprop, Adam (*Adaptive Moment Estimation*), AdaMax, Nadam (versione di Adam che utilizza il momento di Nesterov) e Ftrl (*Follow the regularized leader*) e in tutti sono stati lasciati i parametri di default. Nel confrontare i risultati ottenuti notiamo innanzitutto che SGD e Ftrl hanno performato in maniera peggiore rispetto agli altri, in quanto le loro prestazioni dipendono maggiormente dalla selezione dei parametri. Successivamente notiamo che RMSprop raggiunge un ottimo valore di sensitivity, ma al contempo risulta essere il peggiore in termini di specificity, quindi permette di evitare falsi negativi con alte probabilità ma non è invece in grado di evitare falsi positivi con efficacia. Al contrario sono Nadam e Adamax a raggiungere i valori più alti di specificity e quindi ad aver la minor probabilità di creare "falsi allarmi" classificando come violenti video che invece non contengono scene di violenza. Infine sono

Nadam e Adam ad avere le migliori performance in termini di accuratezza ed F1-score e si è deciso di proseguire con gli esperimenti con l'utilizzo di Adam.

Tabella 3.1: Esperimenti Optimizer

Optimizer	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
SGD	42.6	0.9395	0.9017	0.9580	0.9552
RMSprop	37	0.9469	0.8621	0.9882	0.9616
Adam	30.8	0.9559	0.9086	0.9790	0.9676
Adamax	39.8	0.9517	0.9207	0.9668	0.9642
Nadam	33.6	0.9605	0.9293	0.9756	0.9707
Ftrl	42.2	0.9229	0.8672	0.9500	0.9431

3.3.3 Layer Fully-Connected

Gli esperimenti con i layer fully-connected sono stati effettuati su tre diverse configurazioni:

1. Un unico layer, fc7, che prende in input le feature dal livello fc6 della rete C3D

2. La configurazione precedente con in aggiunta un secondo layer fc8

3. Una configurazione formata da due layer, fc6 e fc7, che prendono in input le feature direttamente dal quinto gruppo di layer convoluzionali della rete C3D.

Dai risultati ottenuti con la prima configurazione si può evincere come al crescere del numero di neuroni, fino ad arrivare a 2048, le prestazioni migliorino ma soltanto leggermente. Nel singolo esperimento eseguito sulla seconda configurazione possiamo invece notare come aggiungere ulteriori layer al nostro classificatore tende a peggiorare le prestazioni del modello. Infine i dati riportati nella tabella mostrano il chiaro aumento di performance portato dall'utilizzo della terza configurazione. Quest'ultima potrebbe raggiungere risultati migliori in quanto prelevare gli input dalla C3D ad un livello precedente consente di ottenere informazioni più "grezze" e di carattere generale, le quali risultano quindi più facilmente riconducibili a delle classi completamente diverse da quelle usate nell'addestramento della C3D. Avendo ottenuto i valori più alti di accuratezza e F1-score, negli esperimenti successivi verrà utilizzata la terza configurazione con un primo layer da 1024 neuroni e un secondo layer da 256.

Tabella 3.2: Esperimenti Layer Fully Connected

FC Layer (Units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
fc7 (256)	29.2	0.9528	0.8897	0.9836	0.9656
fc7 (512)	30.8	0.9559	0.9086	0.9790	0.9676
fc7 (1024)	25.4	0.9559	0.9000	0.9832	0.9677
fc7 (2048)	27.2	0.9588	0.9026	0.9861	0.9699
fc7 (512), fc8 (512)	38.2	0.9517	0.8966	0.9788	0.9646
fc6 (1024), fc7 (256)	17	0.9740	0.9690	0.9765	0.9806
fc6 (2048), fc7 (512)	11.6	0.9692	0.9466	0.9803	0.9773

3.3.4 Dropout

Una delle tecniche usate per far sì che una rete mantenga elevate capacità di generalizzazione è chiamata *Dropout* e consiste nell'ignorare durante l'addestramento alcuni output selezionati casualmente da un determinato layer. Ciò permette di avere un layer con connessioni sia in input sia in output diverse ad ogni iterazione, simulando così un addestramento su più architetture in parallelo. Quindi tramite questa tecnica è possibile impedire che alcuni neuroni finiscano con l'essere addestrati per correggere errori commessi da altri nodi della rete, rendendo così il modello più robusto in quanto maggiormente in grado di comportarsi correttamente di fronte a dati mai visti. Dagli esperimenti sembrerebbe che ad un valore di dropout rate di 0.2 sul layer fc7 corrisponda un maggiore valore di specificity e le prestazioni migliori in termini di accuratezza e F1-score sono state ottenute accoppiando tale valore ad un dropout rate di 0.5 sul layer fc6.

Tabella 3.3: Esperimenti Dropout

Dropout	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
drop1 = 0.5, drop2 = 0.5	17	0.9740	0.9690	0.9765	0.9806
drop1 = 0.5, drop2 = 0.2	19.2	0.9751	0.9655	0.9798	0.9815
drop1 = 0.2, drop2 = 0.5	15	0.9726	0.9405	0.9882	0.9798
drop1 = 0.3, drop2 = 0.3	17	0.9718	0.9466	0.9840	0.9791
drop1 = 0.2, drop2 = 0.2	12.8	0.9715	0.9526	0.9807	0.9788
drop1 = 0.4, drop2 = 0.2	13	0.9726	0.9638	0.9769	0.9795
drop1 = 0.3, drop2 = 0.2	16.2	0.9743	0.9500	0.9861	0.9810

3.3.5 Batch Size

Il batch size determina il numero di elementi esaminati dalla rete prima che l'algoritmo di ottimizzazione effettui una stima dell'errore commesso e aggiorni quindi i pesi. General-

mente grazie a valori alti l'algoritmo di ottimizzazione può ottenere stime maggiormente precise, mentre grazie a valori bassi si possono ottenere modelli più robusti. Dai risultati degli esperimenti si può notare come il valore della sensitivity aumenta con il crescere del batch size, ma considerando accuratezza e F1-score le prestazioni migliori sono state ottenute usando batch di 32 elementi.

Tabella 3.4: Esperimenti Batch Size

Batch Size	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
8	16.2	0.9692	0.9466	0.9803	0.9773
16	18.4	0.9698	0.9509	0.9790	0.9776
32	19.2	0.9751	0.9655	0.9798	0.9815
64	30	0.9743	0.9543	0.9840	0.9809

Capitolo 4

Esperimenti con Reti Neurali Ricorrenti

Nella seconda parte di questo lavoro di tesi sono stati utilizzati dei modelli di deep learning pre-addestrati su database ImageNet facilmente importabili dalla libreria Keras, a cui sono state fatte seguire delle reti neurali ricorrenti. Anche in questo caso è stato usato il dataset AirtLab per testare l'accuratezza delle reti.

4.1 Strumenti Usati

Per quanto riguarda le tecnologie utilizzate in questo capitolo, oltre a quelle descritte nella sezione 3.1, compaiono le reti neurali ricorrenti e le Keras applications.

4.1.1 Reti Neurali Ricorrenti

Al fine di gestire meglio sequenze di dati sono state sviluppate delle reti dette ricorrenti, dove, contrariamente a quanto avviene nelle reti neurali feed-forward, l'output corrente non dipende solo dall'ingresso ma anche dallo stato del sistema. Per addestrare le RNN è necessario utilizzare la retropropagazione nel tempo, come mostrato nella Figura 4.1.

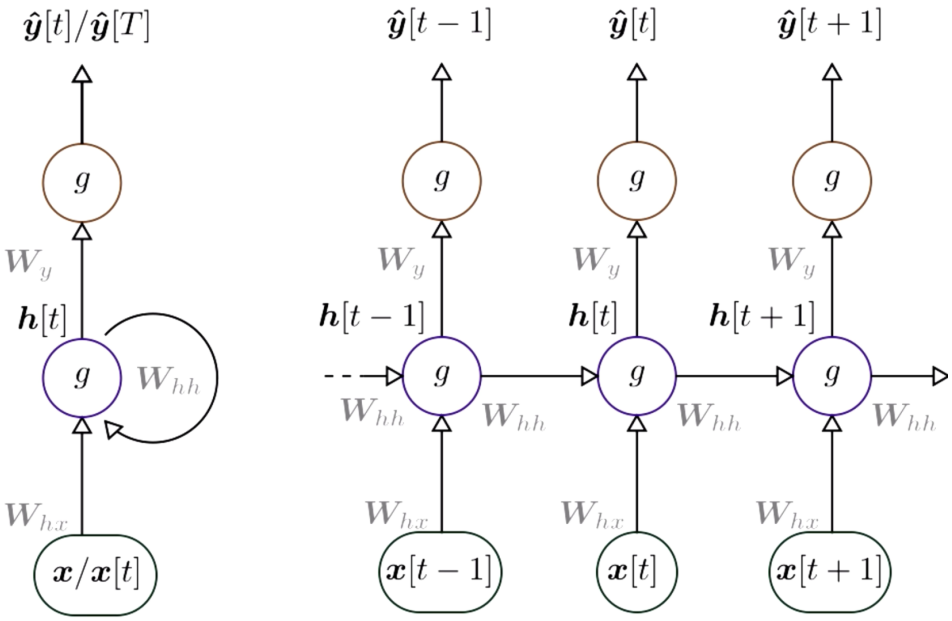


Figura 4.1: Schematizzazione di una rete neurale ricorrente

Tra i maggiori problemi che affliggono questo tipo di reti troviamo la scomparsa e l'esplosione del gradiente. Infatti se per la propagazione si usano delle matrici (rappresentate dalle frecce presenti nella Figura 4.2), dato che queste ultime possono modificare le dimensioni degli output, è possibile che il gradiente aumenti nel tempo fino ad "esplodere" o al contrario il gradiente potrebbe diminuire fino a "scomparire". Nelle normali reti ricorrenti i gradienti vengono propagati attraverso tutte le possibili "frecce" quindi le probabilità di scomparire o esplodere sono alte.

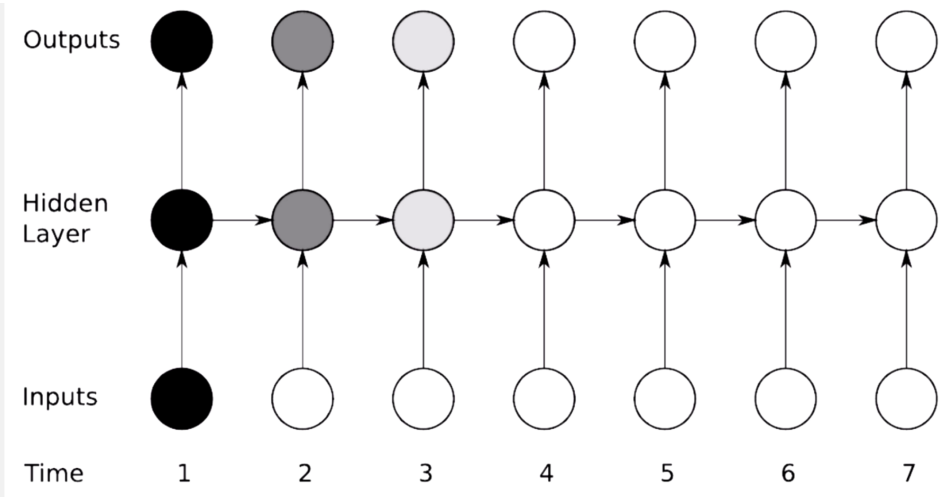


Figura 4.2: Scomparsa del gradiente

Al fine di evitare questi due fenomeni si è pensato di realizzare reti in cui non tutte le

connessioni vengono utilizzate. Nella Figura 4.3 è riportato un esempio di *gated recurrent network*, in cui ogni strato nascosto ha 3 connessioni, che consentono la propagazione del gradiente solamente in alcuni casi, che in figura sono rappresentati da un cerchio.

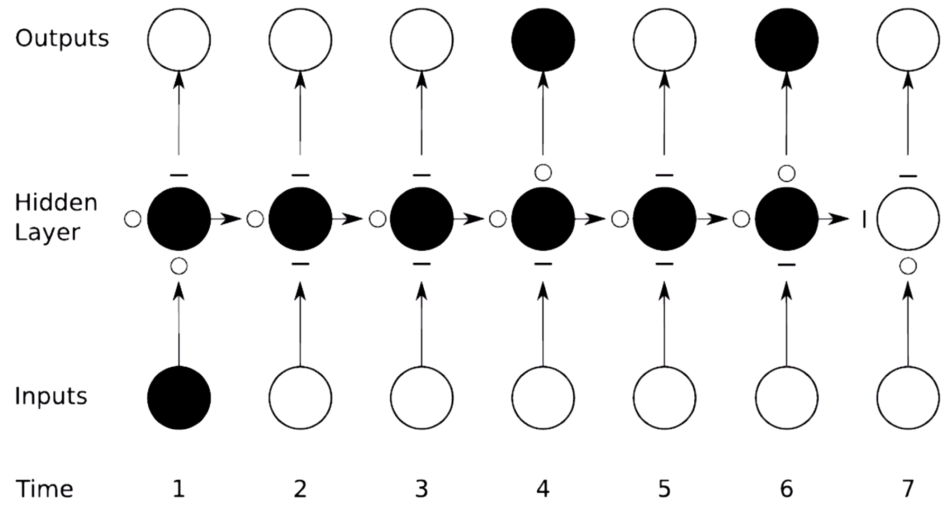


Figura 4.3: Propagazione del gradiente in una gated recurrent network

Tra le RNN che fanno uso di porte logiche troviamo le reti Long Short-Term Memory (LSTM). In queste reti l'output in un dato istante dipende dallo stato della cella, dall'output dell'istante precedente, o stato nascosto precedente, e dall'input dell'istante corrente. La prima porta di cui è dotata una LSTM è chiamata *forget gate*, questa prende in ingresso lo stato nascosto precedente e i dati di input, su cui applica la funzione sigmoide, i cui risultati, che sono compresi tra 0 e 1, vengono poi moltiplicati per lo stato precedente. Ciò significa che alle informazioni irrilevanti corrisponde un risultato della funzione sigmoide prossimo a 0, in quanto ciò porta all'eliminazione di tali informazioni dallo stato della cella. Gli input usati dal forget gate raggiungono poi l'*input gate*, dove viene applicata ancora una volta la funzione sigmoide per decidere quali informazioni sono importanti e quali invece trascurabili, infatti sugli stessi input viene applicata la funzione *tanh* che restituisce invece valori compresi tra -1 e 1 e i risultati ottenuti dalle due funzioni vengono moltiplicati tra loro, successivamente viene eseguita la somma con lo stato della cella per aggiornarla con le nuove informazioni ritenute importanti. Infine è presente l'*output gate* che decide quale deve essere lo stato nascosto. Anche in questa porta logica viene applicata la funzione sigmoide sul precedente stato nascosto e sull'input corrente, dopodiché il risultato viene moltiplicato ai valori ottenuti applicando la funzione *tanh* allo stato attuale della cella per decidere quali informazioni devono essere trasportate dallo stato nascosto. Vista la capacità di mantenere uno stato con le informazioni sugli elementi di input precedenti, su queste reti si basano ad esempio le applicazioni allo stato dell'arte nei campi di *speech recognition*, *speech synthesis* e *natural language understanding*.

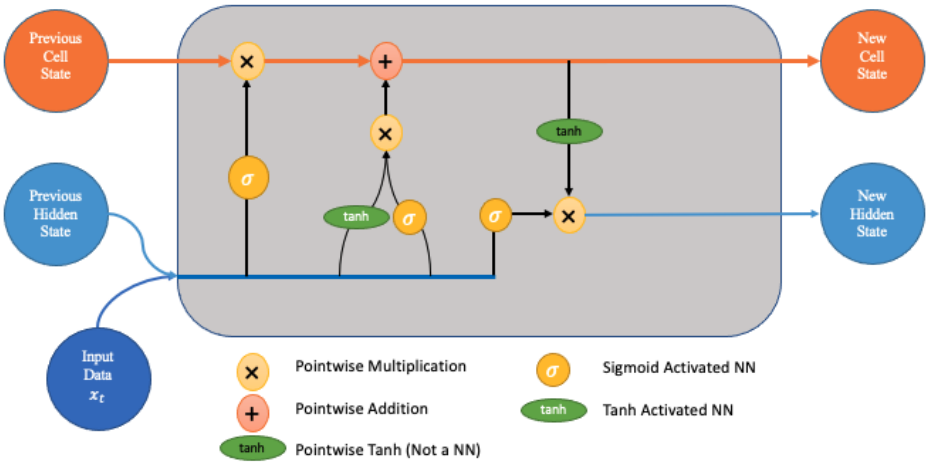


Figura 4.4: Schema di una cella LSTM

A partire dalle LSTM sono state poi sviluppate delle varianti come le Bidirectional LSTM, abbreviate in BiLSTM, le quali sono fondamentalmente formate da due normali LSTM: una in cui l'input viene propagato dall'inizio alla fine della rete, mentre nell'altra viene propagato in senso contrario. L'obiettivo delle BiLSTM è quello di rendere disponibile alla rete ancora più informazione e una maggiore conoscenza del contesto.

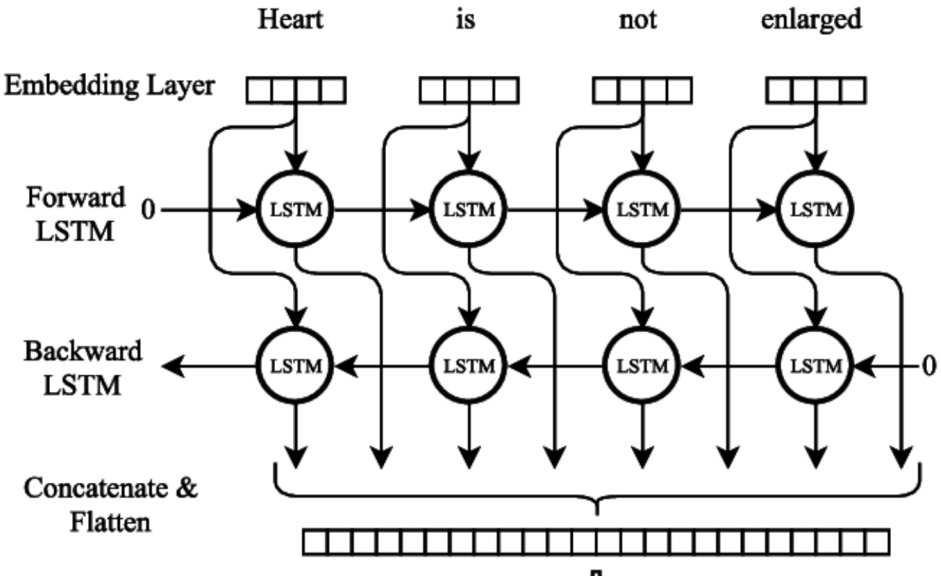


Figura 4.5: Schema di una rete BiLSTM

Mentre per i casi in cui è necessario lavorare con delle immagini si sono uniti i concetti di LSTM e di rete convoluzionale per ottenere le reti ConvLSTM che fanno di uso celle LSTM in cui le operazioni di moltiplicazione tra matrici sono state sostituite da operazione di convoluzione.

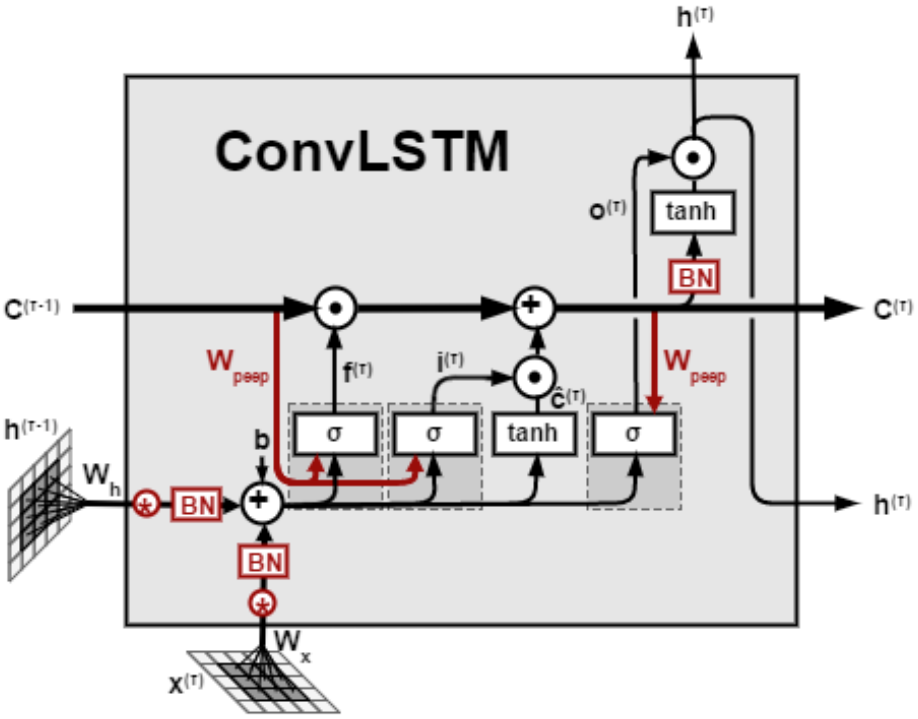


Figura 4.6: Schema di una cella ConvLSTM

4.1.2 Keras Applications

In Keras Applications, la libreria mette a disposizione delle note reti neurali, con la possibilità opzionale di includere i pesi ottenuti durante l'addestramento con il dataset ImageNet, il quale contiene 14 milioni di immagini con 100 classi annotate ed è stato realizzato per l'utilizzo nel campo del riconoscimento di oggetti [29]. Le reti utilizzate in questo lavoro sono le seguenti:

- VGG16, una rete per il riconoscimento di immagini composta da 16 layer convoluzionali e 2 layer fully-connected di 4096 neuroni [17]
- VGG19, versione a 19 layer convoluzionali della rete VGG [17]
- ResNet50V2, una rete composta da 48 layer convoluzionali basata sul concetto di residual learning [30]
- Xception, una rete della famiglia Inception e basata sul concetto di "depthwise separable convolutions" [31]
- Nasnet, basata sul Neural Architecture Search, ovvero un algoritmo che cerca l'architettura in grado di ottenere i migliori risultati in un particolare problema [32]

- InceptionResNetV2, un'architettura che incorpora nelle architetture della famiglia Inception il concetto di residual learning [33].

4.2 Realizzazione

Il sistema di riconoscimento utilizzato in questo capitolo può essere suddiviso nelle seguenti parti:

- Preprocessing dell'input
- ConvNet importata da Keras Applications
- Layer ricorrenti
- Layer fully-connected

Per quanto riguarda il preprocessing dell'input il procedimento seguito è analogo a quello descritto nella sezione 3.2.1 con l'unica differenza che in questo caso le sequenze di frame e le relative etichette non sono state salvate in delle memmap.

4.2.1 ConvNet importata da Keras Applications

Attraverso le reti convoluzionali pre-addestrate con dataset ImageNet è possibile effettuare l'estrazione delle feature presenti nei video a cui la rete viene sottoposta. Purtroppo a causa delle limitazioni imposte dall'istanza gratuita di Colab sull'utilizzo dello storage non è stato possibile utilizzare frame di dimensioni 224x224, che è il formato su cui tutti i modelli sono stati pre-addestrati, bensì le dimensioni dell'input sono state impostate a 112x112x3 come nel capitolo 3.

```
def getVGG16Model(verbose=True):
    vgg = VGG16(include_top=False, weights="imagenet", input_shape=(112,112,3));
    if verbose:
        vgg.summary()
    for layer in vgg.layers:
        layer.trainable = False;
    return vgg

def getVGG19Model(verbose=True):
    vgg = VGG19(include_top=False, weights="imagenet", input_shape=(112,112,3));
    if verbose:
        vgg.summary()
    for layer in vgg.layers:
```

```
        layer.trainable = False;
    return vgg

def getResNet50V2Model(verbose=True):
    resnet = ResNet50V2(include_top=False, weights="imagenet", input_shape
        =(112,112,3));
    if verbose:
        resnet.summary()
    for layer in resnet.layers:
        layer.trainable = False;
    return resnet

def getXceptionModel(verbose=True):
    xcep = Xception(include_top=False, weights="imagenet", input_shape=(112,112,3));
    if verbose:
        xcep.summary()
    for layer in xcep.layers:
        layer.trainable = False;
    return xcep

def getNasnetModel(verbose=True):
    nasn = NASNetMobile(include_top=False, weights="imagenet", input_shape
        =(112,112,3));
    if verbose:
        nasn.summary()
    for layer in nasn.layers:
        layer.trainable = False;
    return nasn

def getInceptionResNetV2(verbose=True):
    incres = InceptionResNetV2(include_top=False, weights="imagenet", input_shape
        =(112,112,3));
    if verbose:
        incres.summary()
    for layer in incres.layers:
        layer.trainable = False;
    return incres
```

4.2.2 Definizione del modello

Alle reti convoluzionali devono poi seguire un layer tra LSTM, BiLSTM o ConvLSTM, successivamente segue un layer fully-connected composto da 256 neuroni e infine viene usato un layer sigmoide di un singolo neurone per la classificazione. A tale scopo viene creato un modello in cui viene inizialmente istanziato il modello convoluzionale all'interno di un *wrapper TimeDistributed*, che fa sì che uno alla volta i 16 frame delle clip vengano presentati alla rete senza che questa cambi i propri parametri tra frame della stessa clip.

```
def getLSTMModel(getConvModel, verbose=True):
    model = Sequential()
    model.add(TimeDistributed(getConvModel(verbose), input_shape=(16,112,112,3)))
```

Dopodiché viene aggiunto uno tra i layer ricorrenti:

- LSTM

```
model.add(TimeDistributed(Flatten()))
model.add(LSTM(units=128, return_sequences=False))
```

- BiLSTM

```
model.add(TimeDistributed(Flatten()))
model.add(Bidirectional(LSTM(units=128, return_sequences=False)))
```

- ConvLSTM

```
model.add(ConvLSTM2D(filters=64, kernel_size=(3,3)))
```

Infine troviamo i layer fully-connected per la classificazione:

```
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
     if verbose:
        model.summary()
    opt = tensorflow.keras.optimizers.Adam()
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

4.3 Esperimenti

Per quanto riguarda l'implementazione della funzione per realizzare gli esperimenti sono state fatte due modifiche rispetto a quella riportata nella sezione 3.2.4. La prima consiste nell'aver impostato come parametro monitorato dall'early stopping il valore della funzione di loss calcolato durante la fase di validazione. Mentre la seconda riguarda la creazione e l'utilizzo di una classe denominata DataGen per la creazione dei batch di training, validazione e test.

```
class DataGen(Sequence) :

    def __init__(self, base_path, filenames, labels, batch_size) :
        self.base_path = base_path
        self.filenames = filenames
        self.labels = labels
        self.batch_size = batch_size

    def __len__(self) :
        return (np.ceil(len(self.filenames) / float(self.batch_size))).astype(np.int
            )

    def __getitem__(self, idx) :
        batch_x = self.filenames[idx * self.batch_size : (idx+1) * self.batch_size]
        batch_y = self.labels[idx * self.batch_size : (idx+1) * self.batch_size]

        return np.array([np.load(os.path.join(self.base_path, file_name)) for
            file_name in batch_x]), np.array(batch_y)
```

4.3.1 Esperimenti sulle reti ricorrenti

Durante l'esecuzione di questi esperimenti si sono osservati due problemi: il primo riguarda l'abbassamento delle performance derivante dall'utilizzo di frame di dimensioni 112x112 invece che 224x224, il secondo invece riguarda l'underfitting di tutti i modelli fatta eccezione di quelli in cui vengono usate VGG16 e VGG19. In particolare, dalle tabelle 4.1 e 4.2, i cui dati relativi all'input di dimensioni maggiori sono stati ottenuti eseguendo gli esperimenti su un'istanza Colab Pro, si può notare come, tra il caso in cui vengono usati fotogrammi 224x224 e quello in cui i frame sono invece 112x112, c'è una differenza tra i valori di accuratezza che raggiunge l'8%, mentre tra i valori di F1-score raggiunge il 5,8%.

Il problema dell'underfitting è invece dovuto al fatto che i parametri su cui si basa la classificazione non sono sufficienti a permettere al modello di comprendere quale sia

Tabella 4.1: Esperimenti VGG16 + ConvLSTM (64)

Input shape	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
112x112x3	7.6	0.8723	0.7147	0.9492	0.9093
224x224x3	12.6	0.9562	0.9181	0.9748	0.9677

Tabella 4.2: Esperimenti VGG19 + ConvLSTM (64)

Input shape	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
112x112x3	17.0	0.8994	0.8578	0.9197	0.9248
224x224x3	12.4	0.9545	0.9302	0.9664	0.9662

la relazione tra dati di input e la loro classe. Come possiamo vedere nella Figura 4.2 il modello classifica tutti i video come violenti in quanto non è riuscito ad identificare alcun confine decisionale nel set di dati.

```
confusion matrix split 1
[[  0 232]
 [  0 476]]
              precision    recall  f1-score   support

 non-violent       0.00      0.00      0.00       232
     violent       0.67      1.00      0.80       476

    accuracy                           0.67       708
   macro avg       0.34      0.50      0.40       708
weighted avg       0.45      0.67      0.54       708

Loss: 0.6326491832733154
Accuracy: 0.6723163723945618
```

Figura 4.7: Split di un esperimento eseguito su un modello affetto da underfitting

Per quanto riguarda gli esperimenti svolti con VGG16 e VGG19 possiamo notare come le prestazioni dei modelli crescono al crescere del numero unità presenti nei layer ricorrenti, inoltre notiamo come la semplice LSTM superi in termini di performance BiLSTM e ConvLSTM.

Tabella 4.3: Esperimenti Reti Ricorrenti con VGG16

Layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
ConvLSTM (64)	7.6	0.8723	0.7147	0.9492	0.9093
ConvLSTM (128)	11.0	0.8825	0.7776	0.9336	0.9144
LSTM (128)	12.0	0.9082	0.8345	0.9441	0.9325
LSTM (256)	11.0	0.9113	0.8362	0.9479	0.9351
LSTM (512)	10.6	0.9127	0.8629	0.9370	0.9352
BiLSTM (128)	14.2	0.9121	0.8940	0.9580	0.9535
BiLSTM (256)	13.8	0.9144	0.8647	0.9387	0.9365

Tabella 4.4: Esperimenti Reti Ricorrenti con VGG19

Layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
ConvLSTM (64)	17.0	0.8994	0.8578	0.9197	0.9248
ConvLSTM (128)	12.8	0.9054	0.8595	0.9277	0.9295
LSTM (128)	16.8	0.9195	0.8448	0.9559	0.9411
LSTM (256)	10.0	0.8992	0.8448	0.9256	0.9250
LSTM (512)	21.8	0.9370	0.8940	0.9580	0.9535
BiLSTM (128)	15.6	0.9127	0.8336	0.9513	0.9362
BiLSTM (256)	15.8	0.9229	0.8853	0.9412	0.9425

4.3.2 Esperimenti con layer Fully Connected

Successivamente sono state verificate le prestazioni ottenibili con le configurazioni in cui alle ConvNet seguono immediatamente uno o due layer fully-connected. Dai risultati ottenuti possiamo concludere che, come avveniva anche nell'esperimento 3.3.3, aggiungere un secondo livello va ad inficiare sulle prestazioni, inoltre notiamo come anche con i layer fc le performance della rete crescono all'aumentare del numero di neuroni. Infine dal confronto tra questi risultati con quelli relativi all'utilizzo delle RNN risultano evidenti le migliori prestazioni dei layer fully-connected a livello di sensitivity e specificity, nonché a livello di F1-score.

Tabella 4.5: Esperimenti Dense Layer con VGG16

Layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
fc1 (256)	7.8	0.9215	0.8172	0.9722	0.9435
fc1 (512)	10.4	0.9280	0.8319	0.9748	0.9482
fc1 (1024)	10.0	0.9384	0.8672	0.9731	0.9555
fc1 (2048)	8.0	0.9599	0.9259	0.9765	0.9704
fc1 (1024), fc2 (256)	8.6	0.9167	0.8147	0.9664	0.9401
fc1 (2048), fc2 (512)	11.4	0.9395	0.8931	0.9622	0.9556

Tabella 4.6: Esperimenti Dense Layer con VGG19

Layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
fc1 (256)	7.4	0.9243	0.8371	0.9668	0.9451
fc1 (512)	5.8	0.9480	0.8957	0.9735	0.9619
fc1 (1024)	8.2	0.9571	0.9224	0.9739	0.9682
fc1 (2048)	8.6	0.9582	0.9259	0.9739	0.9691
fc1 (1024), fc2 (256)	9.6	0.9243	0.8216	0.9743	0.9454
fc1 (2048), fc2 (512)	13.6	0.9398	0.8767	0.9706	0.9559

Esperimenti con Dropout Rate

Gli esperimenti sul dropout rate sono stati svolti sia sul modello in cui viene utilizzato un layer fc con 1024 neuroni sia sul modello in cui lo stesso layer conta invece 2048 neuroni.

Dalle tabelle 4.7 e 4.8 notiamo come nel primo caso le prestazioni risultino migliori per valori bassi del dropout rate, infatti i risultati migliori in termini di accuratezza e F1-score sono stati ottenuti con un dropout rate di 0,2.

Tabella 4.7: Esperimenti Dropout Rate con VGG16 + Dense (1024)

Dropout Rate	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
0.2	7.6	0.9607	0.9200	0.9807	0.9711
0.3	5.2	0.9545	0.9172	0.9727	0.9664
0.4	7.2	0.9585	0.9259	0.9585	0.9693
0.5	10.0	0.9384	0.8672	0.9731	0.9555

Mentre per quanto riguarda il modello con layer fc da 2048 neuroni, le tabelle 4.9 e 4.10 evidenziano come i valori più alti di accuratezza e F1-score sono state ottenuti con il dropout rate impostato a 0,5. Gli esperimenti successivi sono stati svolti impostando nel layer fc il numero di neuroni a 1024 e il dropout rate a 0.2.

Tabella 4.8: Esperimenti Dropout Rate con VGG19 + Dense (1024)

Dropout Rate	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
0.2	8.6	0.9633	0.9310	0.9790	0.9729
0.3	7.0	0.9613	0.9147	0.9840	0.9716
0.4	8.4	0.9480	0.8940	0.9744	0.9619
0.5	8.2	0.9571	0.9224	0.9739	0.9682

Tabella 4.9: Esperimenti Dropout Rate con VGG16 + Dense (2048)

Dropout Rate	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
0.2	4.6	0.9500	0.9319	0.9588	0.9626
0.3	4.4	0.9503	0.9034	0.9731	0.9633
0.4	7.2	0.9444	0.8957	0.9681	0.9591
0.5	8.0	0.9599	0.9259	0.9765	0.9704

Tabella 4.10: Esperimenti Dropout Rate con VGG19 + Dense (2048)

Dropout Rate	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
0.2	9.0	0.9576	0.9207	0.9756	0.9687
0.3	4.0	0.9480	0.9267	0.9584	0.9612
0.4	5.2	0.9517	0.8948	0.9794	0.9647
0.5	8.6	0.9582	0.9259	0.9739	0.9691

Esperimenti con Ottimizzatori

In seguito sono stati testati gli ottimizzatori che avevano meglio performato negli esperimenti 3.3.2, ovvero Adam, Adamax, Nadam e RMSprop. Ancora una volta le prestazioni migliori in termini di accuratezza e F1-score sono state raggiunte da Adam, mentre in termini di specificity Adamax e RMSprop ottengono valori leggermente superiori.

Tabella 4.11: Esperimenti Optimizer con VGG16

Optimizer	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
RMSprop	7.2	0.9571	0.9336	0.9685	0.9681
Adam	7.6	0.9607	0.9200	0.9807	0.9711
Adamax	11.2	0.9613	0.9276	0.9777	0.9715
Nadam	5.4	0.9421	0.9034	0.9609	0.9571

Tabella 4.12: Esperimenti Optimizer con VGG19

Optimizer	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
RMSprop	8.0	0.9571	0.9388	0.9660	0.9679
Adam	8.6	0.9633	0.9310	0.9790	0.9729
Adamax	6.8	0.9582	0.9414	0.9664	0.9688
Nadam	3.4	0.9511	0.9181	0.9672	0.9638

Esperimenti con Batch Size

Infine per quanto riguarda gli esperimenti sul batch size si può concludere che le prestazioni migliori vengono raggiunte utilizzando batch di 16 elementi, tranne per quanto riguarda la sensitivity che raggiunge il valore più alto con l'utilizzo di 8 elementi per batch.

Tabella 4.13: Esperimenti Batch Size con VGG16

Batch Size	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
8	7.6	0.9607	0.9200	0.9807	0.9711
16	7.8	0.9647	0.9483	0.9727	0.9737
32	6.6	0.9636	0.9457	0.9723	0.9729

Tabella 4.14: Esperimenti Batch Size con VGG19

Batch Size	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
8	8.6	0.9633	0.9310	0.9790	0.9729
16	11.0	0.9638	0.9621	0.9647	0.9729
32	6.6	0.9636	0.9457	0.9723	0.9729

Capitolo 5

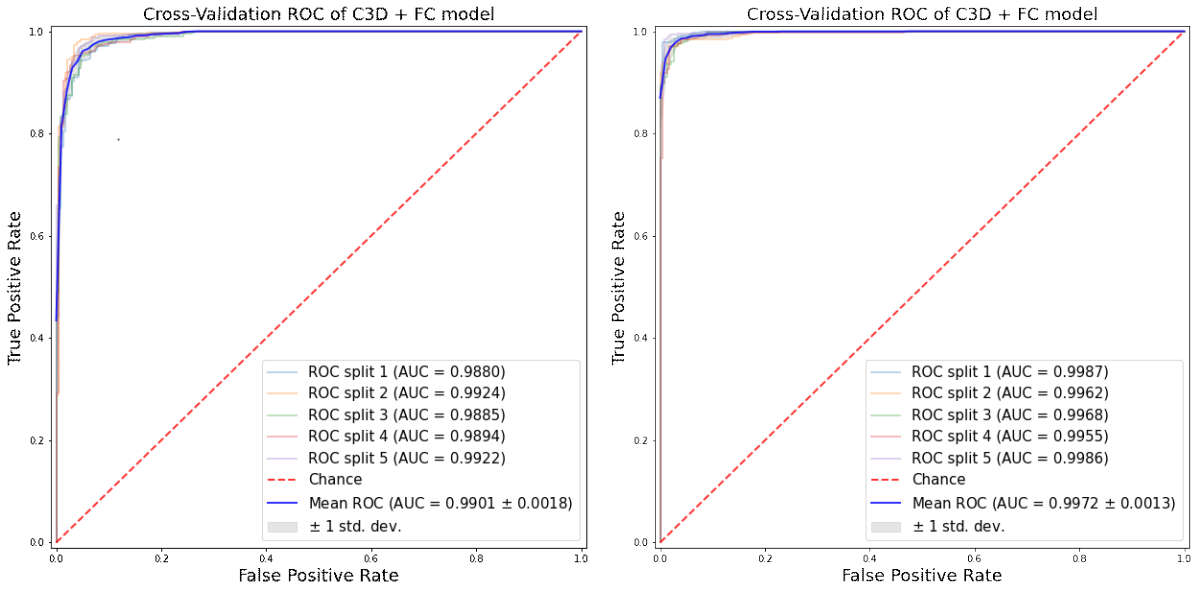
Conclusioni

Nella prima parte di questo lavoro di tesi è stato sviluppato un modello utilizzando la rete C3D pre-addestrata su dataset Sport-1M fino al layer "fc6" per estrarre le feature dai video, le quali sono poi passate ad uno strato fully-connected di 512 neuroni e infine ad un singolo neurone che applicando la funzione sigmoide si occupa della classificazione in video violenti e non violenti. Nel corso degli esperimenti, oltre agli iperparametri, l'architettura stessa del modello ha subito delle modifiche al fine di aumentare le prestazioni fino ad arrivare alla versione finale, in cui viene usata per estrarre le feature la rete C3D fino all'ultimo gruppo di layer convoluzionali, il cui output viene fornito in ingresso a due livelli fully-connected di rispettivamente 1024 e 256 neuroni. Nel modello finale inoltre sono stati impostati i dropout rate dei layer fully-connected rispettivamente a 0.5 e 0.2, il batch size a 32 e come ottimizzatore è stato utilizzato Adam.

Tabella 5.1: Risultati esperimenti con reti neurali feed-forward

Versione modello	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
iniziale	30.8	0.9559	0.9086	0.9790	0.9676
finale	19.2	0.9751	0.9655	0.9798	0.9815

Di seguito viene riportato un confronto tra le AUCROC, ovvero le aree sotto le curve ROC, con cui è possibile visualizzare meglio il rapporto tra falsi positivi e falsi negativi ottenuti nei vari split degli esperimenti.



(a) AUC modello iniziale

(b) AUC modello finale

Mentre nella seconda parte di questo lavoro sono stati utilizzati dei modelli di deep learning pre-addestrati su dataset ImageNet messi a disposizione dalla libreria Keras a cui sono stati fatti seguire delle reti neurali ricorrenti. Purtroppo solo le reti VGG16 e VGG19 non hanno sofferto del fenomeno dell'underfitting e inoltre a causa delle limitazioni dell'istanza gratuita di Colab non è stato possibile verificare quanto migliori sarebbero le performance ottenibili con le reti ricorrenti se si usassero in input dei frame di dimensioni 224x224.

Tabella 5.2: Risultati esperimenti con reti ricorrenti e VGG16

Tipo layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
ConvLSTM (128)	11.0	0.8825	0.7776	0.9336	0.9144
LSTM (512)	10.6	0.9127	0.8629	0.9370	0.9352
BiLSTM (256)	13.8	0.9144	0.8647	0.9387	0.9365

Tabella 5.3: Risultati esperimenti con reti ricorrenti e VGG19

Tipo layer (units)	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
ConvLSTM (128)	12.8	0.9054	0.8595	0.9277	0.9248
LSTM (512)	21.8	0.9370	0.8940	0.9580	0.9535
BiLSTM (256)	15.8	0.9229	0.8853	0.9412	0.9425

Infine si è deciso di eliminare il layer ricorrente per far seguire immediatamente alle ConvNet i layer fully-connected e al fine di ottenere prestazioni migliori si è svolto un

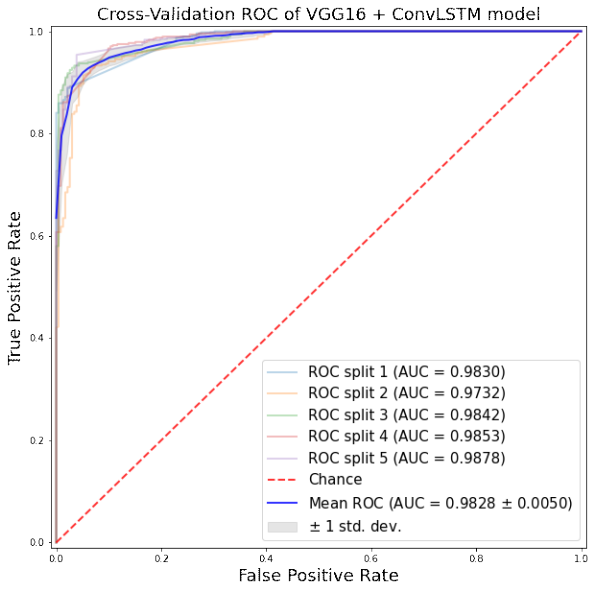
processo di ottimizzazione degli iperparametri come nella prima parte. I modelli finali sono composti da un solo layer fully-connected di 1024 neuroni con un dropout rate di 0.2, fatto seguire alle VGG. Inoltre sono stati utilizzati batch di 16 elementi e Adam come ottimizzatore.

Tabella 5.4: Risultati esperimenti con VGG e layer fc

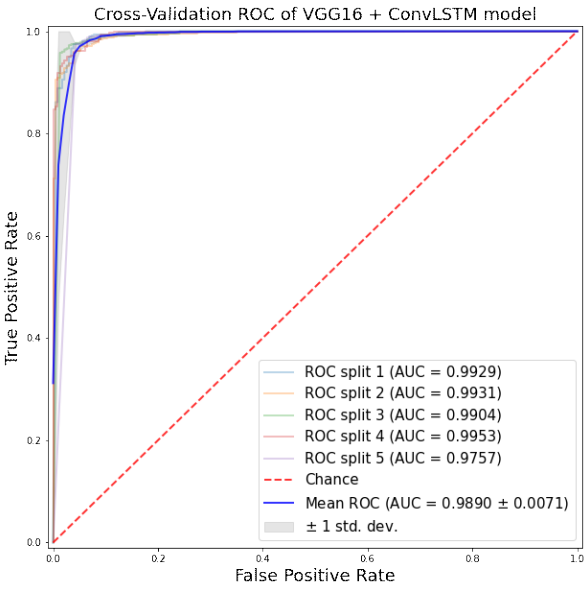
Versione modello	Avg Epoch	Avg Accuracy	Specificity	Sensitivity	F1-score
iniziale (VGG16)	7.8	0.9215	0.8172	0.9722	0.9435
finale (VGG16)	7.8	0.9647	0.9483	0.9727	0.9737
iniziale (VGG19)	7.4	0.9243	0.8371	0.9668	0.9451
finale (VGG19)	11.0	0.9638	0.9621	0.9647	0.9729

Sviluppi futuri:

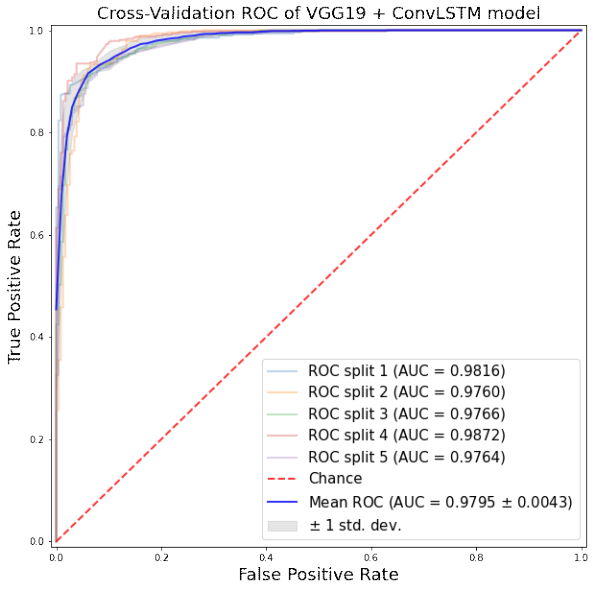
- Eseguire nuovamente gli esperimenti discussi nella sezione 4.3 utilizzando però input di dimensioni 224x224x3 per verificare quali prestazioni è possibile raggiungere utilizzando le reti ricorrenti e i modelli pre-addestrati presenti nella libreria Keras.
- Testare i modelli utilizzati in questo lavoro anche su altri dataset, come ad esempio l'Hockey Fight Dataset e il Crowd Violence Dataset, per poterli confrontare con gli approcci usati nello stato dell'arte.
- Ricercare ulteriori modelli da sostituire alla C3D e alle Keras Applications che permettano di raggiungere performance ancora migliori.



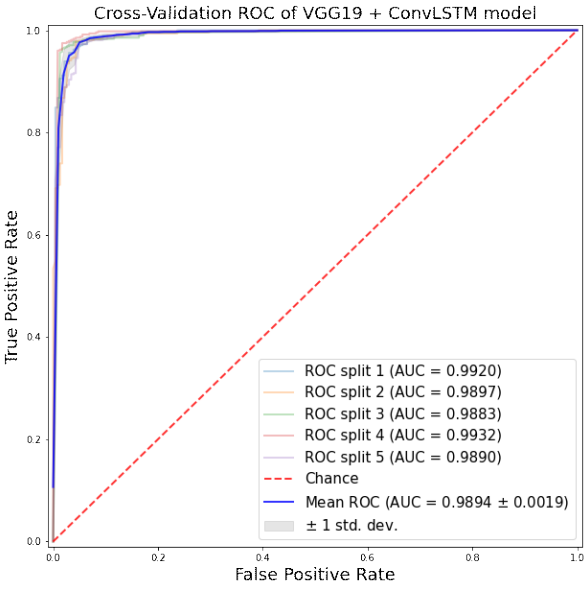
(a) AUC modello iniziale - VGG16



(b) AUC modello finale - VGG16



(c) AUC modello iniziale - VGG19



(d) AUC modello finale - VGG19

Bibliografia

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

[2] Long Xu, Chen Gong, Jie Yang, Qiang Wu, and Lixiu Yao. Violent video detection based on mosift feature and sparse coding. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3538–3542, 2014.

[3] 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.

[4] Ming Yang, Shuiwang Ji, W. Xu, Jinjun Wang, Fengjun Lv, Kai Yu, Yihong Gong, M. Dikmen, D. Lin, and Thomas S. Huang. Detecting human actions in surveillance videos. In *TRECVID*, 2009.

[5] 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.

[6] Zhihong Dong, Jie Qin, and Yunhong Wang. Multi-stream deep networks for person to person violence detection in videos. In Tieniu Tan, Xuelong Li, Xilin Chen, Jie Zhou, Jian Yang, and Hong Cheng, editors, *Pattern Recognition*, pages 517–531, Singapore, 2016. Springer Singapore.

[7] Peipei Zhou, Qinghai Ding, Haibo Luo, and Xinglin Hou. Violent interaction detection in video based on deep learning. *Journal of Physics: Conference Series*, 844:012044, jun 2017.

[8] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning deep representations of appearance and motion for anomalous event detection. *CoRR*, abs/1510.01553, 2015.

[9] 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, Nov 2016.

[10] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2010.

[11] Sarah Almeida Carneiro, Gabriel Pellegrino da Silva, Silvio Jamil F. Guimaraes, and Helio Pedrini. Fight detection in video sequences based on multi-stream convolutional neural networks. In *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 8–15, 2019.

[12] 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*, 19(11), 2019.

[13] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[16] University of Central Florida Center for Research in Computer Vision. Ucf101 - action recognition data set. [Online], 2012. Available: https://www.crcv.ucf.edu/data/UCF101.php [Consultato il 27/08/2021].

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[18] Joe Yue-Hei Ng, Matthew J Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. corr abs/1503.08909 (2015), 2015.

[19] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016.

[20] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *arXiv preprint arXiv:1406.2199*, 2014.

[21] Google colaboratory. [Online], Available: https://colab.research.google.com [Consultato il 27/08/2021].

[22] Project jupyter. [Online], Available: https://jupyter.org [Consultato il 27/08/2021].

[23] Keras. [Online], Available: https://keras.io [Consultato il 27/08/2021].

[24] Opencv. [Online], Available: https://opencv.org [Consultato il 27/08/2021].

[25] Scikit-learn. [Online], Available: https://scikit-learn.org/stable [Consultato il 27/08/2021].

[26] Airtlab dataset. [Online], Available: https://github.com/airtlab/A-Dataset-for-Automatic-Violence-Detection-in-Videos [Consultato il 27/08/2021].

[27] Miriana Bianculli, Nicola Falcionelli, Paolo Sernani, Selene Tomassini, Paolo Contardo, Mara Lombardi, and Aldo Franco Dragoni. A dataset for automatic violence detection in videos. *Data in Brief*, 33:106587, 2020.

[28] C3d repository. [Online], Available: https://github.com/aslucki/C3D_Sport1M_keras [Consultato il 27/08/2021].

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[31] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

[33] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.