UNIVERSITÀ
POLITECNICA
DELLE MARCHE

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

Valutazione degli effetti della rimozione dello sfondo nell'ambito del riconoscimento di scene di violenza attraverso l'utilizzo di reti neurali profonde

Evaluating the impact of background removal on Violence Detection in videos through Deep Neural Networks

Candidato:
PAOLINI DAVIDE

Relatore:
Prof. DRAGONI ALDO FRANCO

Correlatore:
Prof. SERNANI PAOLO

Anno Accademico 2022-2023

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

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

Valutazione degli effetti della rimozione dello sfondo nell'ambito del riconoscimento di scene di violenza attraverso l'utilizzo di reti neurali profonde

Evaluating the impact of background removal on Violence Detection in videos through Deep Neural Networks

Candidato:
PAOLINI DAVIDE

Relatore:
Prof. DRAGONI ALDO FRANCO

Correlatore:
Prof. SERNANI PAOLO

Anno Accademico 2022-2023

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ringraziamenti

Un grazie speciale alla mia famiglia per avermi sempre sostenuto e supportato.

Ringrazio Alessandro e Andrea per la compagnia e l'aiuto che mi hanno offerto in questi due anni.

Grazie al prof. Aldo Franco Dragoni e al prof. Paolo Sernani per i consigli e per avermi dato la possibilità di svolgere sia la tesi triennale sia la tesi magistrale su questo tema così interessante, importante ed attuale.

Ancona, Febbraio 2024

PAOLINI DAVIDE

Sommario

La diffusione sempre più ampia di telecamere di sorveglianza e di fenomeni violenti nella nostra società rende di fondamentale importanza il problema della "Violence Detection". Infatti, la sorveglianza manuale di un elevato numero di telecamere presenta molte difficoltà, a causa della limitata capacità di monitoraggio continuo e della suscettibilità alle distrazioni degli operatori umani. Di conseguenza, si stanno sviluppando delle reti neurali profonde per identificare automaticamente scene violente nei video, ma al momento questi sistemi non raggiungono delle prestazioni accettabili su dati diversi da quelli usati durante l'addestramento. In questo lavoro di tesi viene presentata una soluzione basata sulla rimozione dello sfondo dai fotogrammi che compongono i video e attraverso l'utilizzo di questo approccio viene valutata l'importanza che assume il contesto in fase di addestramento e l'impatto che la sua rimozione ha sulle prestazioni e sulla capacità dei modelli di generalizzare quanto appreso su dati nuovi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La diffusione delle telecamere di videosorveglianza nel mondo è in continua crescita e si stima che attualmente ci siano più di un miliardo di dispositivi installati globalmente [2]. Questo significa che è in aumento anche il carico di lavoro degli operatori umani incaricati di monitorare costantemente le scene riprese da tali dispositivi. Essendo gli operatori soggetti a distrazioni e affaticamento, l'efficacia di questo sistema nel rilevare situazioni critiche può essere facilmente compromessa.

Per affrontare questa sfida, l'implementazione di un sistema automatico di rilevazione delle scene di violenza emerge come una soluzione potenzialmente efficace. Infatti, tale sistema potrebbe sollevare gli operatori umani dalla responsabilità di dover monitorare costantemente tutte le telecamere, consentendo loro di concentrarsi solo sulle situazioni identificate come anomale dal sistema automatico. Tuttavia, i modelli di intelligenza artificiale attualmente disponibili presentano alcune limitazioni, come la necessità di dataset per l'addestramento estremamente ampi e la loro suscettibilità alle variazioni di prospettiva, illuminazione e sfondo nelle scene monitorate. Questi fattori rendono ancora critica la ricerca e lo sviluppo di modelli più avanzati capaci di generalizzare efficacemente la rilevazione di violenza anche in contesti nuovi.

In questa tesi sarà indagata la capacità di generalizzazione di due modelli di deep learning basati sulle reti C3D e VGG16. Al fine di valutare l'efficacia di tali modelli, verranno impiegate le reti YOLOv8 e Mask R-CNN per la segmentazione delle persone presenti nei fotogrammi dei video. Questo processo permetterà poi di rimuovere lo sfondo dai video, riducendo le possibili interferenze dovute alle variazioni di ambiente, prospettiva e illuminazione.

1.1 Obiettivi della tesi

- Valutare l'impatto della rimozione dello sfondo sulla performance delle reti neurali profonde nel task di rilevazione di violenza nei video.

- Analizzare l'apprendimento delle reti neurali profonde da contesti specifici e valutare in che misura esse si affidino allo sfondo nel rilevare scene di violenza.
- Valutare la capacità di generalizzazione delle reti neurali profonde, verificando le loro prestazioni su video con caratteristiche diverse rispetto a quelli utilizzati nella fase di addestramento.

1.2 Struttura della tesi

Il presente lavoro di tesi è suddiviso nel seguente modo:

- Nel secondo capitolo vengono illustrate soluzioni simili, presenti in letteratura, al problema della scarsa capacità di generalizzazione, applicate sempre nell'ambito della violence detection.
- Nel terzo capitolo vengono descritte le reti convoluzionali, i modelli per la segmentazione delle immagini, i dataset e i vari strumenti che sono stati utilizzati nella realizzazione degli esperimenti. Sarà poi analizzata l'implementazione della funzione dedicata alla rimozione dello sfondo dai fotogrammi che compongono i video e verranno approfondite le strutture dei modelli utilizzati per l'estrazione delle feature e la classificazione dei filmati. Infine, saranno riportati e analizzati i risultati ottenuti negli esperimenti condotti.
- Nel quarto capitolo vengono presentati in modo chiaro tutti i risultati ottenuti, evidenziando osservazioni e considerazioni emerse. Infine, si presentano delle proposte per possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

La rimozione del contesto nell'ambito della Violence Detection è un tema di crescente interesse nella letteratura recente. Infatti, negli ultimi anni si è assistito a un'accelerazione significativa degli studi in questo settore che hanno come obiettivo lo sviluppo di sistemi automatici dotati di una più ampia capacità di generalizzazione.

Nel seguito saranno analizzati tre articoli molto recenti, per comprendere le metodologie utilizzate e i risultati in ciascuno di questi studi.

2.1 Apprendimento non supervisionato

Ehsan et al. [3] propongono una soluzione al problema di rilevazione di violenza nei video basata sull'apprendimento non supervisionato. Con questa scelta cercano di superare le problematiche legate all'insufficiente disponibilità di dataset validi e la variabilità degli ambienti e delle attività umane.

La soluzione proposta (Figura 2.1) fa utilizzo di due flussi, uno contenente le informazioni spaziali e l'altro quelle temporali. Partendo dall'ipotesi che quando le persone combattono i loro corpi assumono una posa specifica, viene utilizzata la rete YOLO [4] per rilevare gli esseri umani presenti nei fotogrammi al fine di generare un flusso spaziale privo di informazioni rumorose o riguardanti gli elementi di sfondo (Figura 2.2). Inoltre, ritenendo l'Optical Flow non abbastanza discriminante, Ehsan et al. [3] hanno introdotto una nuova feature chiamata "Jerk", la quale cattura le variazioni di accelerazione, permettendo alla rete di separare le attività normali, contraddistinte da una "velocità costante", dalle azioni violente.

Le feature vengono poi elaborate attraverso un AutoEncoder (AE) convoluzionale. A differenza di tecniche di compressione dei dati come la Principal Component Analysis (PCA) e Independent Component Analysis (ICA), l'AE può comprimere l'input solo se simile ai dati su cui è stato addestrato. Poiché quest'ultimo impara le caratteristiche delle scene non violente, non riesce a

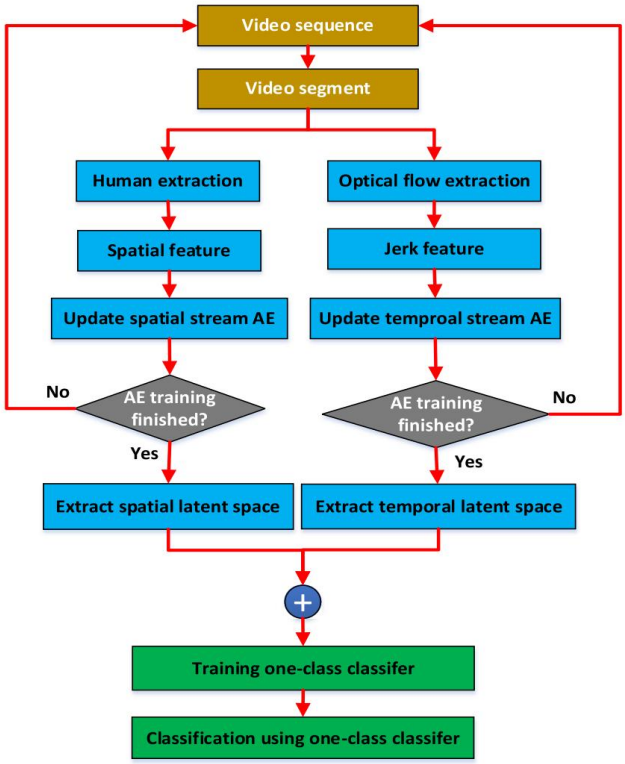


Figura 2.1: Rappresentazione della pipeline proposta da Ehsan et al.

comprimere correttamente le azioni violente, perciò Ehsan et al. [3] partono da questa idea per riuscire a riconoscere le azioni violente.

Per la classificazione viene utilizzato il metodo K-Nearest Neighbors (KNN), il quale crea una regione che rappresenti i dati usati per l'addestramento (che sono solo campioni di azioni normali) e classifica come violenti tutti i campioni di test la cui distanza dai k campioni più vicini è superiore ad una certa soglia.

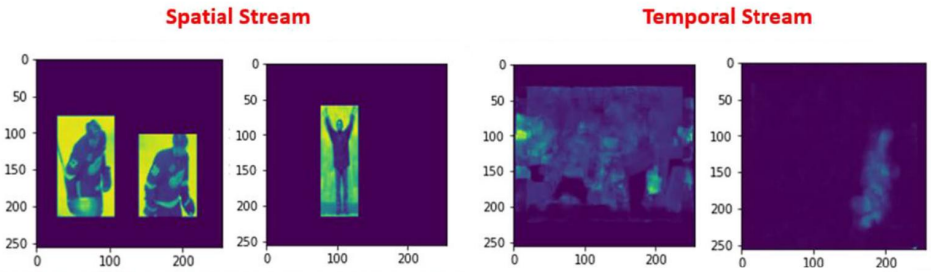


Figura 2.2: Esempi di flusso spaziale e flusso temporale usati da Ehsan et al.

I risultati ottenuti sui dataset Hockey Fight e Movies mostrano come alcuni lavori precedenti riescano a raggiungere prestazioni migliori (Figura 2.3), ma negli esperimenti cross-dataset (Figura 2.4 tutti questi metodi vedono un crollo nell'accuratezza raggiunta, mentre il metodo proposto da Ehsan et al. [3]

mantiene delle ottime performance, dimostrando la sua maggiore capacità di generalizzazione.

Supervision	Method	AUC (%)	
		Hockey	Movies
Supervised	ViF [11]	83	98
	OViF [27]	87	N/A
	Fast fight [28]	76	97
	BoW (STIP) [30]	86	82
	LMP [12]	84	92
	LSTM [31]	95	99
	C3D [33]	93	87
	3D CNN [34]	96	99
Unsupervised	This work (Temporal stream)	80	95
	This work (Spatial stream)	73	89
	This work (Spatio-temporal streams)	84	98

Figura 2.3: Confronto con altri metodi sui dataset Hockey Fight e Movies

Training dataset	Testing dataset	AUC (%)		
		Our method	3D CNN [34]	C3D [33]
Hockey	Movies	97	49	48
Movies	Hockey	71	63	59
Hockey	Hockey	84	96	87
Movies	Movies	98	99	93

Figura 2.4: Risultati degli esperimenti cross-dataset

2.2 Classificazione dei fotogrammi chiave

Bi et al. [5] propongono un sistema di rilevamento della violenza che si basa sull'estrazione dai video dei "fotogrammi chiave" (*keyframes*) e la loro classificazione, con il fine di ottenere un modello robusto e leggero computazionalmente.

L'algoritmo di estrazione dei fotogrammi chiave utilizza il *Frame Difference Method* e seleziona i fotogrammi che hanno il valore massimo all'interno di una finestra scorrevole di N frame consecutivi (Figura 2.5). Questa scelta deriva dal fatto che almeno alcuni dei comportamenti non violenti più comuni (come abbracciarsi o stringersi la mano) possono apparire simili a comportamenti violenti per un breve periodo di tempo, perciò una lunghezza appropriata della finestra può ridurre le probabilità che questi vengano segnalati come violenti. Questa soluzione riduce quindi il numero di fotogrammi elaborati a 1/K del totale.

Successivamente, la rete ResNet18 viene utilizzata per estrarre le feature dai fotogrammi chiave e classificarli come violenti o non violenti. Infine, l'intero

video viene classificato come violento se il numero di fotogrammi chiave violenti contenuti supera il valore soglia impostato (Figura 2.5).

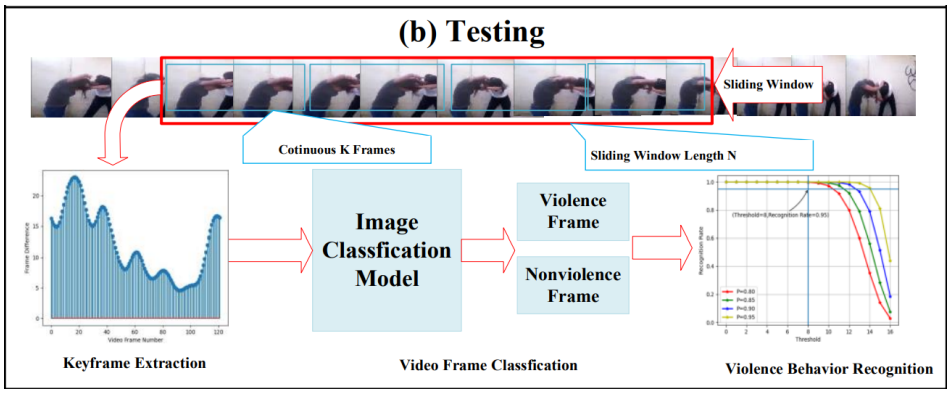


Figura 2.5: Soluzione proposta da Bi et al.

Inoltre, sono stati estratti i fotogrammi chiave dai dataset RLVS [6], RWF-2000 [7] e Hockey Fight [8], e sono state annotate 5000 pose umane, al fine di addestrare Deeplab-V3plus [9], ovvero un modello di segmentazione semantica. Quest'ultimo viene utilizzato per generare la versione segmentata dei frame, con l'obiettivo di rendere il sistema di riconoscimento maggiormente robusto, dandogli in input sia i fotogrammi originali sia quelli segmentati.

2.3 Pose Estimation

L'architettura proposta da Garcia-Cobo et al. [10] è composta da due pipeline principali: una dedicata all'estrazione della posa delle persone presenti nella scena e l'altra volta a stimare le differenze tra i frame nel tempo (Figura 2.6).

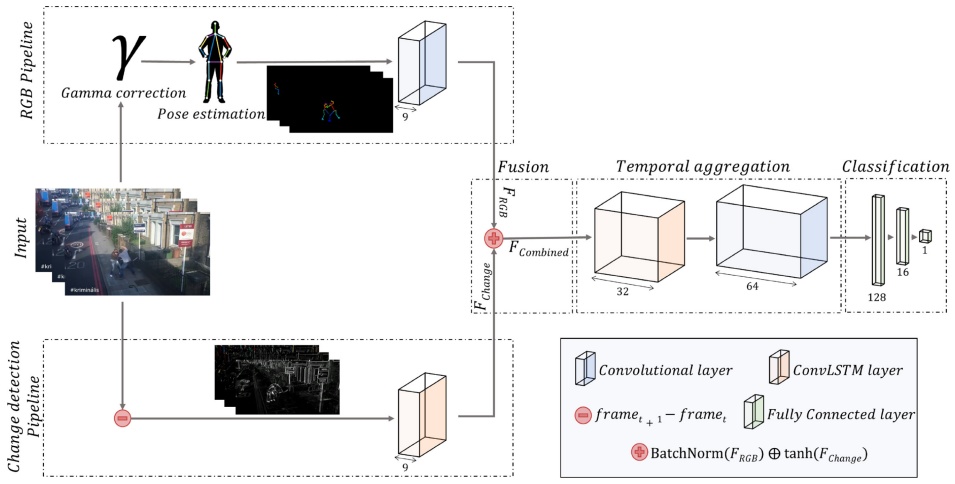


Figura 2.6: Architettura proposta da Garcia-Cobo et al.

Per mantenere il più basso possibile il carico computazionale, nella pipeline RGB sono state utilizzate le predizioni finali degli scheletri, colorando ogni parte con colori specifici, al fine di differenziare le diverse parti del corpo. Inoltre, per eliminare eventuali informazioni rumorose, è stato rimosso lo sfondo dai fotogrammi come mostrato nella Figura 2.7. Infine, per far corrispondere le dimensioni e processare i dati relativi agli scheletri, è stato inserito uno strato convoluzionale.

(a) Skeletons with background (b) Skeletons without background

Figura 2.7: Rimozione dello sfondo dalla pipeline RGB

L'altra pipeline utilizza invece il metodo Frame Difference (differenza tra il frame corrente e il successivo) per catturare il movimento dei soggetti nel tempo. In questa pipeline è stata invece inserita una ConvLSTM, con il fine di fornire più informazioni "inter-frame" possibili.

Dopo aver elaborato l'ultimo frame della sequenza video, la ConvLSTM produce un insieme di feature map. Attraverso una convoluzione Depthwise queste informazioni vengono ulteriormente affinate, prima di raggiungere gli strati fully-connected che operano la classificazione dell'intera sequenza video.

Capitolo 3

Esperimenti

3.1 Strumenti usati

Gli esperimenti sono stati eseguiti sui sistemi di riconoscimento presentati nel lavoro di tesi triennale [11] basati sulle reti convoluzionali C3D e VGG16. Per eseguire la segmentazione dei fotogrammi dei video sono stati scelti due modelli pre-addestrati molto diffusi, ovvero YOLOv8 [4] e Mask R-CNN [12].

3.1.1 C3D

Il modello C3D, sviluppato dai ricercatori di Facebook per il task di action recognition [13], è stato addestrato una prima volta sul dataset UCF101, il quale è un dataset di dimensioni intermedie e contenente 101 classi di azioni [14]. L'architettura della rete è composta da 8 layer convoluzionali, 5 layer max-pooling, 2 layer fully-connected e un layer softmax (Figura 3.1.1). L'input della rete è costituito da frame 3x16xHxW, dove 3 sono i canali RGB, 16 la profondità temporale e HxW le dimensioni dei fotogrammi.

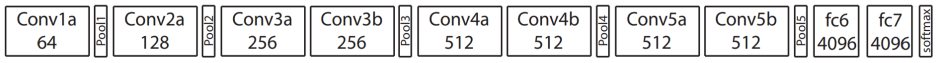


Figura 3.1: Architettura della rete C3D

Un secondo addestramento è stato svolto sul dataset Sport-1M [15], nato dalla collaborazione tra Google e l'università di Stanford, composto da oltre un milione di video sportivi disponibili su YouTube. L'utilizzo di un dataset così ampio favorisce il raggiungimento di prestazioni ottime, contribuendo a migliorare la capacità di generalizzazione della rete e prevenendo il fenomeno dell'overfitting [16] [17]. Per costruire questo dataset sono stati estratti da ciascun video cinque clip di 2 secondi, ridimensionate per ottenere frame 128x171 e successivamente ritagliate per ottenere un input di dimensioni 3x16x112x112.

L'addestramento è stato eseguito con l'algoritmo *Stochastic Gradient Descent* e un learning rate variabile, il quale parte dal valore 0,003 e viene dimezzato ogni 150.000 iterazioni. L'intero processo ha richiesto 1.900.000 iterazioni, equivalenti a circa 13 epoche.

3.1.2 VGG16

La rete VGG16 è stata sviluppata da Simonyan et al. [18] con lo scopo di superare le prestazioni raggiunte da altri sistemi nei compiti di riconoscimento visivo e classificazione delle immagini. Con questa nuova architettura si è voluta sottolineare l'importanza della profondità della rete, dimostrando che l'aumento del numero di strati può portare a una maggiore capacità di apprendimento e, di conseguenza, a migliori prestazioni.

VGG16 è composta da 13 layer convoluzionali, 5 layer max-pooling, e 3 layer fully-connected, seguiti infine da un layer softmax per la classificazione (Figura 3.2).

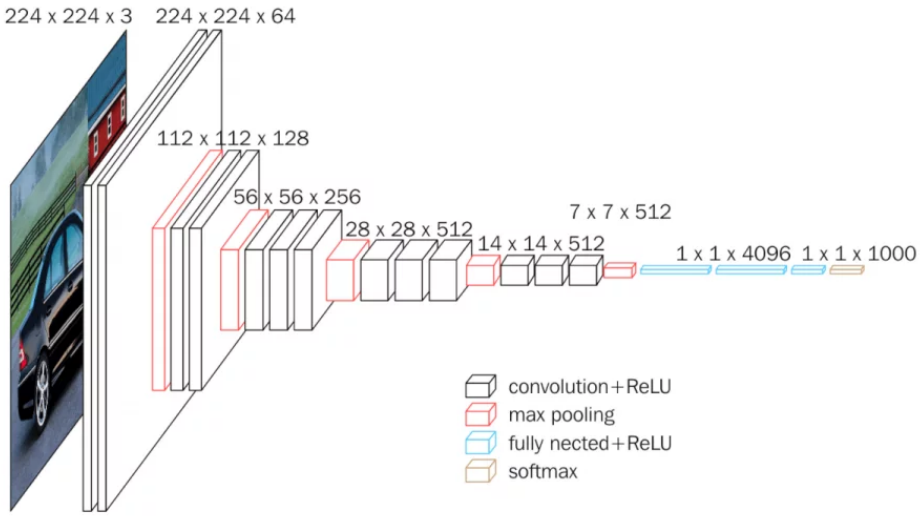


Figura 3.2: Architettura della rete VGG16

Il dataset utilizzato in fase di addestramento è ImageNet, un ampio dataset contenente oltre 14 milioni di immagini e oltre 20.000 classi [19]. L'uso di un dataset così vasto e diversificato ha permesso a VGG16 di ottenere un'ottima capacità di generalizzazione, rendendolo adatto ad operare in una vasta gamma di compiti in ambito Computer Vision. Per l'addestramento, VGG16 utilizza l'algoritmo Stochastic Gradient Descent (SGD) con momentum. Il learning rate iniziale viene impostato a 0,01 e viene ridotto di 10 volte ogni volta che la velocità di apprendimento si stabilizza. Inoltre, vengono applicate altre tecniche come il *weight decay* e il *dropout* per prevenire l'overfitting.

3.1.3 YOLO

Il sistema originale YOLO (acronimo di You Only Look Once), introdotto da Redmon et al. [4], ha ridefinito il task di object detection come un problema di regressione, eseguendo la predizione di bounding boxes e classi direttamente dalle immagini complete in un singolo passaggio, senza la necessità di sfruttare componenti separati per la classificazione e la localizzazione. Infatti, a differenza dei sistemi precedenti, come i Deformable Part Models (DPM) e R-CNN, che adottano approcci basati su finestre o regioni di interesse, YOLO analizza l'intera immagine in un'unica passata. Questo design unificato di YOLO consente di raggiungere tempi di inferenza sensibilmente inferiori, riuscendo però a mantenere anche una precisione media elevata.

Inizialmente, YOLO opera la suddivisione dell'immagine in una griglia di dimensioni SxS. Ogni cella di questa griglia ha la responsabilità di classificare gli oggetti il cui "centro" ricade al suo interno. Inoltre, ciascuna cella genera B bounding box e calcola i relativi *Confidence score*. Questi punteggi sono fondamentali poiché indicano la Intersection Over Union (IOU) tra il bounding box predetto e quello reale (*ground-truth*). L'IOU è calcolato come il rapporto tra l'area dell'intersezione e l'area dell'unione di questi due bounding box. Una limitazione cruciale di YOLO è che ogni cella della griglia è progettata per classificare un singolo oggetto. Di conseguenza, se più oggetti si trovano all'interno della stessa cella la rete non riesce a classificare entrambi correttamente.

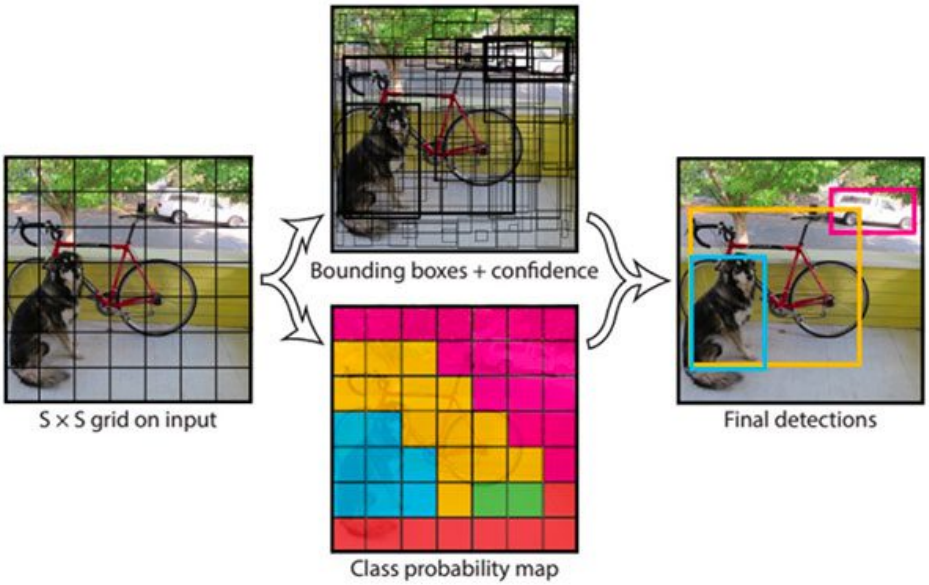


Figura 3.3: Funzionamento di YOLO

Il design della rete è ispirato al modello GoogLeNet [20] per la classificazione

delle immagini e comprende 24 strati convoluzionali seguiti da 2 strati fully-connected. Per quanto riguarda l'addestramento, i primi 20 strati convoluzionali sono stati pre-addestrati sul dataset ImageNet [19], successivamente sono stati aggiunti ulteriori strati per permettere al modello di eseguire il rilevamento degli oggetti nelle immagini, la rete così composta è stata addestrata per circa 135 epoche sui dataset PASCAL VOC 2007 e PASCAL VOC 2012 [21][22], con tecniche come il dropout e la data augmentation per prevenire l'overfitting.

YOLOv8 mantiene il funzionamento base della versione originale, ma introduce diversi miglioramenti sostanziali. Sono attualmente disponibili per l'utilizzo dei modelli pre-addestrati sul dataset COCO [23] per i task di Object Detection, Instance Segmentation e Pose Estimation, oppure pre-addestrati sul dataset ImageNet [19] per i compiti di classificazione.

3.1.4 Mask R-CNN

Mask R-CNN [12] rappresenta un'estensione di Faster R-CNN [24], una tra le architetture più utilizzate nel campo della Computer Vision, specialmente per quanto riguarda i compiti di Object Detection e Instance Segmentation. Faster R-CNN ha introdotto un sistema in due fasi che utilizza una Region Proposal Network (RPN) per generare delle proposte di regioni che contengono un oggetto per poi eseguire la classificazione e raffinare la predizione del bounding box sull'oggetto contenuto nella regione proposta durante la prima fase.

L'architettura di Mask R-CNN può essere suddivisa nelle seguenti componenti fondamentali:

- Backbone Network
- Region Proposal Network
- RoIAlign
- Mask Head

Le backbone usate in Mask R-CNN sono ResNet o ResNeXt, ovvero delle reti neurali convoluzionali pre-addestrate. A questa segue una Feature Pyramid Network (FPN) per creare una *feature pyramid* multi-scala combinando le feature estratte a diversi livelli della backbone. Questa rappresentazione consente al modello di rilevare con precisione oggetti di varie dimensioni presenti nell'immagine.

La componente "ROIAlign" è stata inserita con l'obiettivo di risolvere il problema del disallineamento dovuto al *ROI pooling*. Il suo compito è allineare le caratteristiche all'interno di una regione di interesse (ROI) con la griglia

spaziale della mappa delle feature in output. Infatti, a differenza del ROI pooling che arrotonda le coordinate spaziali al numero intero più vicino, ROIAlign utilizza l'interpolazione bilineare.

Inoltre, Mask R-CNN aggiunge un'ulteriore funzionalità, ovvero delineare anche il contorno preciso attraverso una maschera di segmentazione. Questa capacità è resa possibile dalla "Mask Head", che consiste in un ramo dedicato alla previsione delle maschere per ogni Regione di Interesse (RoI), il quale opera in parallelo con i rami che si occupano della classificazione e della regressione dei box individuati dalla RPN. Questo nuovo ramo consiste di piccole Fully Convolutional Network (FCN), una per ogni RoI, permettendo così la previsione delle maschere pixel-per-pixel, caratteristica distintiva di questo modello.

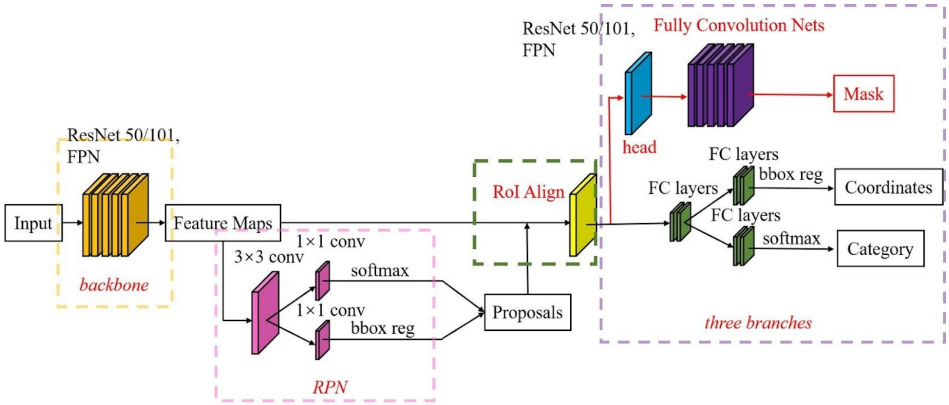


Figura 3.4: Architettura di Mask R-CNN [1]

Mask R-CNN è stato addestrato sulle immagini del dataset COCO [23] ridimensionate per ottenere un lato corto di 800 pixel. Durante l'addestramento, vengono utilizzati mini-batch che comprendono un mix di Regioni di Interesse (RoI), con un rapporto di una RoI positiva (con IoU maggiore di 0,5 rispetto a un ground-truth box) per ogni tre RoI negative. L'efficacia di Mask R-CNN è stata dimostrata attraverso la sua performance superiore nei benchmark del dataset COCO, dove ha superato altre soluzioni allo stato dell'arte sia nella Instance Segmentation che nella Object Detection.

3.1.5 Dataset AirtLab

Il dataset AirtLab [25] è stato progettato per affrontare una sfida importante nel campo della Violence Detection: l'erronea classificazione di comportamenti non violenti come violenti a causa della presenza di movimenti rapidi o altri tipi di contatto fisico somiglianti a comportamenti violenti. Il dataset AirtLab è costituito da 350 clip video (in formato MP4, 1920×1080 pixel, 30 fps), delle

quali 230 sono etichettate come "violent" e 120 come "non-violent". Nei video non violenti sono riprese azioni come abbracci, applausi, esultanze o altri gesti, che sono particolarmente rilevanti per addestrare la rete a non classificarli come falsi positivi. Le scene sono state interpretate da attori non professionisti, i quali variano in numero da 2 a 4 per clip. Un aspetto distintivo di questo dataset è che ogni scena è stata ripresa da due diverse prospettive, utilizzando due diverse telecamere, per fornire una visione olistica dei comportamenti e per testare l'efficacia degli algoritmi di rilevamento della violenza in condizioni variabili.

3.1.6 Dataset Hockey Fight

Il dataset Hockey Fight [8] consiste in una collezione di 1000 clip video della durata di un secondo estratte da partite di hockey, contenente sia momenti di gioco regolare (non violenti) sia episodi violenti. La peculiarità del dataset risiede nel suo contesto dinamico, specifico e caotico, offrendo sfide uniche in termini di movimenti rapidi e occlusioni.

3.1.7 Dataset UCF-Crime

Il dataset UCF-Crime [26] si distingue per la sua ampiezza e per la rappresentazione di eventi reali. Comprendendo un totale di 1900 video di sorveglianza non tagliati e lunghi, UCF-Crime offre un'enorme quantità di dati con 128 ore totali di riprese video, i quali hanno però una risoluzione bassa, pari a 320x240, e una qualità solitamente scadente. Ciò che rende questo dataset unico è la presenza di ben 13 classi di anomalia, ovvero: abuso, arresto, incendio doloso, aggressione, incidente stradale, effrazione, esplosione, lotta, rapina, sparatoria, furto, taccheggio e vandalismo.

Per la raccolta dei video sono stati addestrati dieci meccanismi di annotazione con diversi livelli di competenza nel campo della visione artificiale, per poi prelevare il materiale da piattaforme come YouTube e LiveLeak, effettuando le ricerche in diverse lingue al fine di massimizzare la diversità degli eventi raccolti. Successivamente, i video sono stati accuratamente controllati e filtrati per assicurarsi che fossero autentici e rappresentativi della anomalia desiderata [26].

Il dataset è diviso in un set di addestramento composto da 800 video "normali" e 810 video "anomali", e un set di test contenente 150 video "normali" e 140 video "anomali".

3.1.8 Tecnologie utilizzate

La fase di segmentazione e rimozione dello sfondo dai video, in quanto più leggera computazionalmente, è stata eseguita utilizzando un ambiente Jupyter Notebook [27] su un PC Desktop, equipaggiato con 16GB di RAM e una GPU NVidia RTX 3060Ti.

Per l'addestramento delle reti di classificazione, essendo invece necessarie maggiori risorse, si è scelto di sfruttare Google Colaboratory [28], un servizio cloud che offre ambienti Jupyter Notebook anche gratuitamente. In particolare, sono stati utilizzati dei runtime con "RAM elevata" e GPU V100 per gli esperimenti su singolo dataset, mentre è stato necessario sfruttare le GPU A100 per gli esperimenti cross-dataset.

Il linguaggio di programmazione scelto per questo progetto è Python nella sua versione 3.10.12, noto per l'ampio ecosistema di librerie dedicate al Machine Learning e al Deep Learning. Tra queste, si è deciso di utilizzare:

- Keras [29]
- OpenCV [30]
- NumPy [31]
- Scikit-learn [32]

Keras [29] è una libreria nota per la sua capacità di facilitare lo sviluppo e la valutazione di reti neurali profonde, in quanto offre un alto livello di astrazione, permettendo così di realizzare reti neurali complesse con poche righe di codice.

OpenCV [30], un'altra libreria open source focalizzata sulla Computer Vision e l'Image Processing, è stata impiegata nella fase di preprocessing dei video.

NumPy [31] offre invece supporto per maneggiare array e matrici multidimensionali, mettendo a disposizione una vasta gamma di funzioni per eseguire operazioni su questo tipo di strutture.

Infine, Scikit-learn (abbreviabile in Sklearn) [33] è stata di grande supporto nelle fasi di addestramento delle reti e valutazione delle predizioni. Questa libreria open-source contiene numerose funzioni utili nel campo dell'apprendimento automatico, oltre ad essere altamente compatibile con le altre librerie usate.

3.2 Realizzazione

3.2.1 Creazione del dataset FA

L'ampiezza del dataset UCF-Crime [26] e la presenza di classi di anomalia che non sono caratterizzate da scene di violenza tra due o più soggetti hanno reso necessaria la creazione del dataset "FA", il cui nome deriva dalle iniziali delle classi "Fighting" (lotta) e "Assault" (aggressione) selezionate per la loro maggiore rilevanza nell'ambito della Violence Detection.

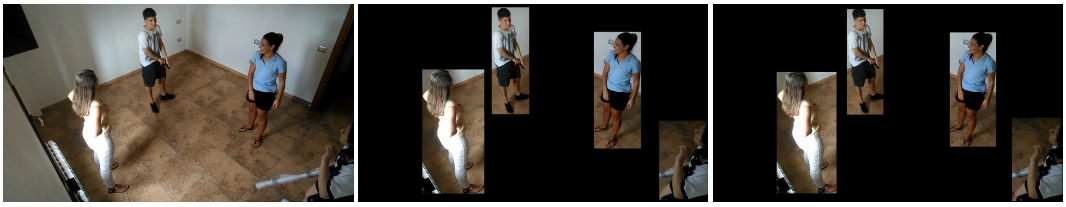
Il primo passo in questa trasformazione è stata appunto la selezione, per la classe "violent", dei soli video appartenenti alle categorie "Fighting" e "Assault". Questo processo ha portato ad un primo filtraggio e quindi alla rimozione di un solo video, denominato "Fighting046_x264.mp4", giudicato scarsamente pertinente. In seguito, sono stati selezionati casualmente un numero simile di video normali, ottenendo così un dataset di 98 video classificati come non violenti e 99 video classificati come violenti.

Per ridurre ulteriormente le dimensioni, è stato intrapreso un lavoro di editing manuale sui video con l'obiettivo di eliminare eventuali scene di "intro" e "outro" e per assicurarsi che le clip ritraessero esclusivamente l'evento violento, o nel caso dei video normali, fossero semplicemente più brevi. Attraverso questo passaggio sono stati eliminati i contenuti superflui e ridotta significativamente la durata complessiva dei video contenuti nel nuovo dataset. Infatti, il dataset FA consiste di 31 minuti e 50 secondi di video violenti e 31 minuti e 48 secondi di video normali.

3.2.2 Rimozione dello sfondo

Per realizzare il processo di rimozione dello sfondo, sono state sviluppate due funzioni specifiche, una per il modello di segmentazione YOLO [4] e una per Mask R-CNN [12]. Entrambe le funzioni operano con lo scopo di modificare i fotogrammi dei video, isolando le figure umane dal resto dell'immagine. Ciò avviene attraverso l'utilizzo dei bounding box o delle maschere prodotte dai modelli di segmentazione, che identificano le regioni del frame in cui sono presenti persone. Una volta ottenute queste informazioni, le funzioni trasformano in nero il colore di tutti i pixel che non rientrano in tali regioni, eliminando così l'ambiente circostante i soggetti ripresi.

In aggiunta, è stata sviluppata una terza funzione che utilizza i fotogrammi modificati per creare una nuova copia del dataset. Ciò avviene salvando i video in una differente directory, mantenendo l'organizzazione originale. La decisione di separare la rimozione dello sfondo dalla fase di preprocessing



(a) Fotogramma originale (b) Segmentato con YOLO (c) Segmentato con Mask R-CNN

Figura 3.5: Esempio dal dataset AirtLab

(a) Fotogramma originale (b) Segmentato con YOLO (c) Segmentato con Mask R-CNN

Figura 3.6: Esempio dal dataset Hockey Fight

dell'input è stata presa per ottimizzare l'uso delle risorse computazionali offerte dall'abbonamento Colab Pro. In questo modo, i video vengono elaborati una sola volta in locale, riducendo così il carico computazionale necessario durante la fase di addestramento e valutazione dei modelli di classificazione sulla piattaforma cloud.

Un aspetto cruciale di questa procedura è la gestione dei fotogrammi in cui i modelli di segmentazione non riescono a rilevare persone. Quindi, è stata introdotta un'opzione nella funzione di salvataggio dei video modificati che permette di scegliere se includere o meno questi fotogrammi (composti da soli pixel di colore nero, dunque privi di informazione) durante la ricostruzione dei video.

3.2.3 Pre-processing dell'input

Nella fase di preprocessing dei video, uno degli aspetti più delicati è la preparazione dei dati in modo che siano compatibili con le esigenze strutturali della rete C3D, che richiede input costituiti da segmenti di 16 frame [13]. Le problematiche principali derivano dal fatto che i video sono di durata variabile e che le scene violente si presentano con discontinuità senza coprire l'intera durata delle registrazioni.

Per affrontare questi problemi, si è adottata una tecnica di suddivisione

(a) Fotogramma originale (b) Bounding box di YOLO (c) Bounding box di Mask R-CNN

Figura 3.7: Esempio dal dataset FA con bounding box

(a) Maschere di YOLO (b) Maschere di Mask R-CNN

Figura 3.8: Esempio dal dataset FA con maschere

dei video, descritta nella tesi di Accattoli [34], che mira a massimizzare la dimensione dell'input e a preservare il maggior numero possibile di informazioni utili. In particolare, i video vengono suddivisi in modo sequenziale in segmenti composti da 16 frame. Ogni segmento così ottenuto eredita l'etichetta del video originale. Inoltre, i fotogrammi vengono ridimensionati per essere di dimensione 112x112 pixel.

Tuttavia, questo procedimento può portare a scartare i frame finali dei video, in quanto la lunghezza totale dei video raramente corrisponde a un multiplo esatto di 16 fotogrammi.

Infine, nel caso in cui verrà utilizzata la rete C3D per l'estrazione delle feature, le sequenze di frame e le relative etichette vengono salvate su due file memmap, al fine di evitare il caricamento contemporaneo in memoria di tutte le sequenze, mentre per gli esperimenti con VGG16 vengono usati dei file ".npy" della libreria NumPy.

3.2.4 Estrazione delle feature

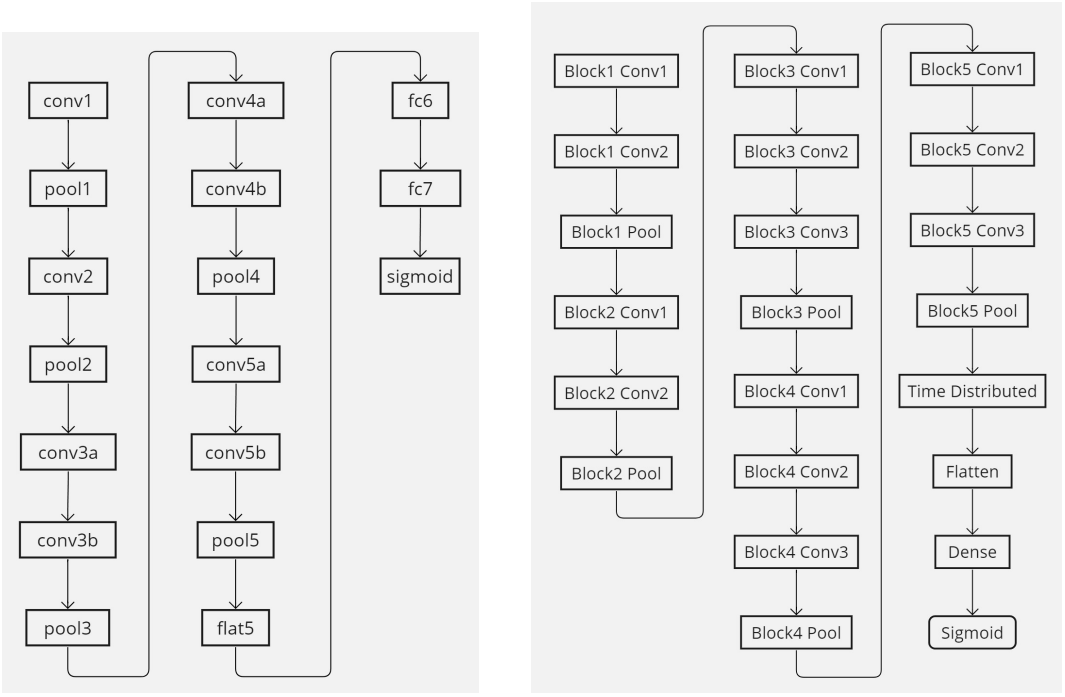
Per la fase di estrazione delle feature dai video sono state impiegate due reti neurali distinte.

La prima rete è la C3D [13], pre-addestrata sul dataset Sport-1M [15], discussa nel capitolo 3.1.1. Si è deciso di estrarre le feature dall'ultimo gruppo di strati convoluzionali, da cui si ottiene un vettore monodimensionale di 8192 elementi che condensa le informazioni provenienti da tutti i layer convoluzionali della rete. Non è stato scelto uno degli strati successivi per evitare di incorporare le elaborazioni specifiche per la classificazione dei video in categorie sportive, che avviene nella parte più profondi della rete. Tale decisione è fondamentale, in quanto le caratteristiche distintive dei diversi sport potrebbero non essere pertinenti o addirittura fuorvianti per il riconoscimento di atti violenti.

La seconda rete è la VGG16 [18], discussa nel capitolo 3.1.2. La rete, pre-addestrata sul dataset ImageNet [19], è resa disponibile dalla libreria Keras Applications. Le dimensioni dell'input sono state ridotte da 224x224x3 a 112x112x3 (che è la dimensione usata anche da C3D), a causa della mancanza di sufficienti risorse computazionali. Analogamente alla C3D, anche per la VGG16, le feature vengono estratte dall'output del quinto gruppo di strati convoluzionali.

3.2.5 Classificazione

Nella fase di classificazione, le feature estratte dalla rete C3D vengono passate a uno strato fully-connected composto da 1024 neuroni. Questo strato ha il



(a) C3D + classificatore (b) VGG16 + classificatore

Figura 3.9: Architetture utilizzate

compito di condensare le informazioni estratte. Successivamente, le informazioni vengono trasmesse a un altro strato denso, questa volta di 256 neuroni, che affina ulteriormente l'elaborazione. Il passaggio finale della classificazione avviene tramite un singolo neurone dotato di funzione di attivazione sigmoide. Questo neurone ha il compito di predire la probabilità con cui i segmenti di video appartengono alle due possibili categorie: "violent" e "non-violent".

Per il modello basato sulla rete VGG16 viene adottata un'architettura simile. In particolare, la rete convoluzionale viene inizialmente incapsulata in un wrapper TimeDistributed della libreria Keras. Questo approccio garantisce che ciascuno dei 16 frame delle clip video venga processato individualmente dalla rete, senza che quest'ultima vari i propri parametri per l'intera durata del segmento. Anche in questo caso, segue uno strato fully-connected di 1024 neuroni, il cui output viene poi classificato da un singolo neurone con funzione di attivazione sigmoide.

3.2.6 Implementazione

Rimozione del contesto

La funzione riportata di seguito fa utilizzo del modello di segmentazione YOLOv8 per rimuovere lo sfondo dal fotogramma passato in chiamata. Innanzitutto viene chiamata la funzione *predict* per eseguire l'inferenza sul frame, chiedendo di rilevare soltanto le persone, ovvero la classe 0.

Se la variabile *use_masks* è *True*, il codice si concentra sull'uso delle maschere. Queste ultime vengono ridimensionate per adattarsi alle dimensioni originali dell'immagine. Infine, si applica la maschera al fotogramma originale per isolare le persone e rimuovere lo sfondo tramite la funzione *cv2.bitwise_and*.

Se *use_masks* è *False*, il codice lavora invece con i bounding box. In questo caso viene creata una maschera (*background_mask*) che inizialmente copre tutta l'immagine e i bounding box vengono utilizzati per "cancellare" le parti della maschera in corrispondenza delle persone. Infine, la maschera così ottenuta viene applicata all'immagine originale per ottenere la versione con lo sfondo rimosso.

Nel caso in cui YOLO non riesca a rilevare persone all'interno dell'immagine la funzione ritorna il valore *None* se *remove_frames* è vero, in caso contrario ritorna un'immagine interamente nera delle stesse dimensioni dell'originale.

```
def remove_context(image):
    results = model.predict(source = image, classes = 0)[0]
    rows, cols = results.orig_shape

    if use_masks is True:
        if results.masks is None:
            if remove_frames:
                return None
            else:
                final_img = np.zeros_like(image)
        else:
            people_masks = results.masks.cpu().numpy().data
            people_mask = np.any(people_masks, axis=0)
            resized_mask = cv2.resize(people_mask.astype(np.uint8), (cols
                ,rows), interpolation=cv2.INTER_NEAREST)
            final_img = cv2.bitwise_and(image, image, mask=resized_mask)
    else:
        if remove_frames is True and boxes.shape[0] == 0:
            return None
```

```
        orig_img = results.orig_img
        background_mask = np.ones((rows, cols), dtype=np.uint8)
        boxes = results.boxes.cpu().numpy()

        for box in boxes.xyxy:
            box = box.astype(int)
            background_mask[box[1]:box[3]+1, box[0]:box[2]+1] = 0

        people_mask = 1 - background_mask
        people_mask_rgb = np.repeat(people_mask[:, :, np.newaxis], 3,
            axis = 2)
        final_img = orig_img * people_mask_rgb

    return final_img
```

Di seguito è riportata la funzione che utilizza invece la rete Mask R-CNN per eseguire la segmentazione dell'immagine. In questo caso è necessaria una pre-elaborazione dell'immagine e l'estrazione esplicita di punteggi, classi, maschere e bounding box. Questi risultati vengono filtrati prima per punteggio, il quale deve superare la soglia impostata a 0.5, e successivamente per classe, la quale è pari a 1 nel caso delle persone. La rimozione dello sfondo avviene in maniera analoga alla precedente funzione, con la sola differenza che le maschere non devono essere ridimensionate "manualmente".

```
def remove_context(image):
    threshold=0.5

    transform = T.Compose([T.ToTensor()])
    image_transform = transform(image).to('cuda')

    pred = model([image_transform])
    pred_score = list(pred[0]['scores'].detach().cpu().numpy())
    pred_t = [pred_score.index(x) for x in pred_score if x>threshold]
    if len(pred_t) > 0:
        pred_t = pred_t[-1]
    else:
        pred_t = -1
    masks = (pred[0]['masks']>0.5).squeeze().detach().cpu().numpy()
    pred_class = list(pred[0]['labels'].cpu().numpy())
    pred_boxes = [[(i[0], i[1]), (i[2], i[3])] for i in list(pred[0]['
```

```
    boxes'].detach().cpu().numpy())]
masks = masks[:pred_t+1]
boxes = pred_boxes[:pred_t+1]
pred_class = pred_class[:pred_t+1]

if use_masks is True:
    people_masks = [masks[i] for i in range(len(masks)) if pred_class
        [i] == 1]

    if remove_frames and len(people_masks) == 0:
        return None
    people_masks = np.any(people_masks, axis=0)
    if type(people_masks) is not np.ndarray or type(people_masks[0])
        is not np.ndarray:
        if remove_frames:
            return None
        else:
            return np.zeros_like(image)
    final_img = np.zeros_like(image)
    final_img[people_masks] = image[people_masks]
else:
    if remove_frames is True and len(people_boxes) == 0:
        return None
    rows, cols, chs = image.shape
    background_mask = np.ones((rows, cols), dtype=np.uint8)
    people_boxes = [boxes[i] for i in range(len(boxes)) if pred_class
        [i] == 1]

    for box in people_boxes:
        background_mask[int(box[0][1]):int(box[1][1])+1, int(box
            [0][0]):int(box[1][0])+1] = 0

    people_mask = 1 - background_mask
    people_mask_rgb = np.repeat(people_mask[:, :, np.newaxis], 3,
        axis = 2)
    final_img = image * people_mask_rgb

return final_img
```

Creazione del modello basato su C3D

La funzione *create_C3D_model* si occupa di definire il modello C3D utilizzando l'API sequenziale di Keras. Viene definita la forma che dovrà avere l'input, ovvero segmenti della lunghezza di 16 frame di dimensione 112x112 e con 3 canali di colore (RGB). Dopo gli strati convoluzionali, che si occupano di estrarre le feature spazio-temporali dai video, viene appiattito l'output (strato *flat5*) per poterlo dare in ingresso al classificatore. Infatti, chiamando la funzione *getFeatureExtractor* con gli opportuni parametri, il modello appena definito viene compilato, selezionando come funzione di loss Mean Squared Error e l'ottimizzatore Stochastic Gradient Descent, vengono caricati i pesi ottenuti dal pre-addestramento sul dataset Sport-1M e infine viene selezionato *flat5* come strato di output. Nella funzione *getC3DCNNModel* viene bloccato l'addestramento della rete C3D e viene definito il modello utilizzato per la classificazione. Questo è composto di due strati fully-connected da 1024 e 256 neuroni, seguiti da uno strato con un unico neurone che esegue la classificazione con la funzione sigmoide. Sono stati inseriti anche degli strati di dropout per prevenire l'overfitting durante l'addestramento. Infine l'intero modello viene compilato, impostando Binary Cross Entropy come funzione di loss e Adam come ottimizzatore.

```
def create_C3D_model():
    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(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'))

    return model
```

```
def getFeatureExtractor(weightsPath, layer):
    model = create_C3D_model()
    model.load_weights(weightsPath)
    model.compile(loss='mean_squared_error', optimizer='sgd')

    return Model(inputs=model.input,outputs=model.get_layer(layer).output
        )

def getC3DCNNModel():
    pretrainedModel = getFeatureExtractor('weights/weights.h5', 'flat5',
```

```
        False)
    for layer in pretrainedModel.layers:
        layer.trainable = False

    dropout0 = Dropout(0)(pretrainedModel.output)
    fc6 = Dense(1024, activation='relu', name='fc6')(dropout0)
    dropout1 = Dropout(.5)(fc6)
    fc7Alt = Dense(256, activation='relu', name='fc7-alt')(dropout1)
    dropout2 = Dropout(.2)(fc7Alt)
    output = Dense(1, activation='sigmoid')(dropout2)

    model = Model(inputs=pretrainedModel.inputs, outputs=output)
    opt = tf.keras.optimizers.Adam()
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['
        accuracy'])

    del pretrainedModel
    return model
```

Creazione del modello basato su VGG16

Le funzioni riportate di seguito si occupano invece di istanziare il modello di classificazione basato sulla rete VGG16. In particolare, la funzione *getVGG16Model* importa il modello VGG16, pre-addestrato sul dataset ImageNet, utilizzando Keras Applications. Gli strati densi finali vengono esclusi impostando a *False* il parametro *include_top* poiché anche in questo caso l'obiettivo è utilizzare questa rete solo per estrarre le feature. Le dimensioni dei frame in input è stato abbassato dal valore di default 224x224 a 112x112 per diminuire il carico computazionale. Analogamente a quanto fatto con C3D, tutti gli strati della rete VGG16 vengono impostati come non addestrabili. Nella funzione *getClassifier* viene invece definito il modello di classificazione con l'API sequenziale di Keras, partendo dall'utilizzo del wrapper TimeDistributed per poter processare sequenze di 16 frame. A questo seguono un primo strato fully-connected da 1024 neuroni e uno strato finale da un unico neurone con funzione di attivazione sigmoide. Come visto nelle funzioni sopra, viene usato il dropout per prevenire il fenomeno dell'overfitting e il modello viene compilato impostando la Binary Cross Entropy come funzione di loss e Adam come ottimizzatore.

```
def getVGG16Model():
```

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

def getClassifier():
    model = Sequential()

    model.add(TimeDistributed(getVGG16Model(), input_shape=(16,112,112,3)
        ))
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))

    opt = tensorflow.keras.optimizers.Adam()
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['
        accuracy'])

    return model
```

Addestramento e Testing

La funzione che si occupa di eseguire addestramento e test dei modelli sopra riportati carica i dati utilizzando le *memmap* (o i file ".npy"), create in fase di pre-processing, al fine di non dover caricare il dataset nella memoria RAM. Successivamente, viene utilizzato il meccanismo di cross-validation *StratifiedShuffleSplit* a 4 split per mantenere la stessa proporzione tra video violenti e non violenti nei set di addestramento e nei set di test. Per ogni split, vengono selezionati casualmente degli indici dal set di addestramento per creare un set di validazione, ottenendo così una suddivisione dei dati 60%-15%-25% destinati rispettivamente all'addestramento, alla validazione e al testing. Quindi, vengono creati dei *DataGenerator* per ciascun set, al fine di facilitare l'addestramento in batch da 32 elementi senza dover caricare tutti i dati in memoria. Il modello viene creato tramite una delle due funzioni descritte in precedenza (passata come parametro *getModel*) e viene addestrato per un massimo di 50 epoche. Infatti, è stato introdotto anche un meccanismo di *Early Stopping* per interrompere l'addestramento nel caso in cui la loss di validazione

non dovesse migliorare per più di 5 epoche consecutive. Infine, il modello viene valutato sul set di test e vengono calcolate le varie metriche di valutazione delle prestazioni del modello.

```
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 = 4
    cv = StratifiedShuffleSplit(n_splits=nsplits, train_size=0.75,
        random_state = rState)
    i = 1

    for train, test in cv.split(X, y):
        num_train_samples = len(train)
        validation_split = 0.20
        num_validation_samples = int(num_train_samples * validation_split
            )
        np.random.seed(42)
        validation_indices = np.random.choice(num_train_samples,
            num_validation_samples, replace = False)
        remaining_indices = np.setdiff1d(np.arange(num_train_samples),
            validation_indices)
        validation_indices = train[validation_indices]
        remaining_indices = train[remaining_indices]

        X_train = np.memmap(os.path.join(mmapDatasetBasePath, '
            samples_train.mmap'), mode='w+', dtype=np.float32, shape=X[
            remaining_indices].shape)
        X_val = np.memmap(os.path.join(mmapDatasetBasePath, 'samples_val.
            mmap'), mode='w+', dtype=np.float32, shape=X[
            validation_indices].shape)
        X_test = np.memmap(os.path.join(mmapDatasetBasePath, '
            samples_test.mmap'), mode='w+', dtype=np.float32, shape=X[test
            ].shape)
```

```
X_train[:] = X[remaining_indices][:]
y_train = y[remaining_indices][:]
train_gen = DataGenerator(X_train, y_train, batchSize)
X_val = X[validation_indices][:]
y_val = y[validation_indices][:]
val_gen = DataGenerator(X_val, y_val, batchSize)
X_test[:] = X[test][:]
y_test = y[test][:]
test_gen = DataGenerator(X_test, y_test, batchSize)

model = getModel(i==1)

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

model.fit(train_gen, validation_data=val_gen, epochs=50, verbose
    =1, callbacks=[es])

evaluation = model.evaluate(test_gen)
scores.append(evaluation)
probs = model.predict(test_gen, verbose=1).ravel()

...
```

3.3 Esperimenti

3.3.1 Metriche di valutazione

Come nel precedente lavoro di tesi [11], verrà data grande importanza alle metriche di accuratezza e *F1-score*, ma saranno tenute in considerazione anche *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, una sensitivity bassa si

traduce in un'elevata probabilità che il sistema non lanci un allarme in caso di evento violento; mentre ad un basso valore di specificity corrisponde una grande quantità di falsi allarmi. Infine, 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 Risultati ottenuti con il dataset AirtLab

Da questi primi risultati possiamo notare come i due sistemi di rilevazione utilizzati rispondono alla rimozione dello sfondo in maniera molto simile. Infatti, in entrambi i casi si ha una diminuzione importante delle prestazioni e la metrica che risente maggiormente della rimozione del contesto è la Specificity.

Tabella 3.1: Esperimenti su dataset AirtLab

CNN	ISM	Frame senza detection	Spec.	Sens.	F1-score	Accuracy
C3D			**0,9474**	**0,9824**	**0,9784**	**0,9709**
	YOLO	Presenti	0,8422	0,9625	0,9439	0,9231
	YOLO	Rimossi	0,8207	0,9613	0,9384	0,9152
	MRCNN	Presenti	0,8698	0,9492	0,9433	0,9232
	MRCNN	Rimossi	0,8802	0,9381	0,9398	0,9191
VGG16			**0,9310**	**0,9738**	**0,9702**	**0,9598**
	YOLO	Presenti	0,8234	0,9182	0,9161	0,8871
	YOLO	Rimossi	0,8352	0,9185	0,9189	0,8912
	MRCNN	Presenti	0,8152	0,9311	0,9213	0,8931
	MRCNN	Rimossi	0,8393	0,9226	0,9221	0,8952

3.3.3 Risultati ottenuti con il dataset Hockey Fight

Gli esperimenti eseguiti con il dataset Hockey Fight l'impatto della rimozione dello sfondo risulta meno evidente rispetto al caso precedente.

Tabella 3.2: Esperimenti su Hockey Fight

CNN	ISM	Frame senza detection	Spec.	Sens.	F1-score	Accuracy
C3D			**0,9620**	**0,9891**	**0,9760**	**0,9756**
	YOLO	Presenti	0,9550	0,9107	0,9315	0,9328
	YOLO	Rimossi	0,9442	0,9542	0,9503	0,9493
	MRCNN	Presenti	0,9450	0,9484	0,9470	0,9467
	MRCNN	Rimossi	0,9592	0,9520	0,9558	0,9556
VGG16			**0,9392**	**0,9571**	**0,9482**	**0,9490**
	YOLO	Presenti	**0,9392**	0,9135	0,9254	0,9263
	YOLO	Rimossi	0,9193	0,9367	0,9297	0,9281
	MRCNN	Presenti	0,9384	0,9294	0,9337	0,9339
	MRCNN	Rimossi	0,8906	0,9440	0,9205	0,9176

3.3.4 Risultati ottenuti con il dataset FA

I risultati ottenuti con il dataset FA evidenziano maggiormente come i frame privi di detection abbiano un effetto negativo sulle prestazioni, soprattutto se si utilizza la rete YOLOv8 per la segmentazione dei fotogrammi. L'utilizzo di quest'ultima porta anche a performance sensibilmente peggiori di quelle ottenute usando Mask R-CNN, probabilmente a causa del gran numero di soggetti (anche non partecipanti all'atto violento) presenti in alcune scene e di occlusioni che si verificano nelle riprese effettuate da vere telecamere di videosorveglianza.

Tabella 3.3: Esperimenti su dataset FA

CNN	ISM	Frame senza detection	Spec.	Sens.	F1-score	Accuracy
C3D			**0,9997**	**0,9989**	**0,9993**	**0,9993**
	YOLO	Presenti	0,9621	0,8664	0,9098	0,9143
	YOLO	Rimossi	0,9609	0,9708	0,9679	0,9661
	MRCNN	Presenti	0,9719	0,9442	0,9576	0,9580
	MRCNN	Rimossi	0,9650	0,9846	0,9764	0,9752
	YOLO (Mask)	Presenti	0,9659	0,8672	0,9124	0,9162
	YOLO (Mask)	Rimossi	0,9587	0,9610	0,9619	0,9599
	MRCNN (Mask)	Presenti	0,9728	0,9428	0,9573	0,9577
	MRCNN (Mask)	Rimossi	0,9830	0,9758	0,9799	0,9793
VGG16			**0,9986**	**0,9977**	**0,9982**	**0,9982**
	YOLO	Presenti	0,9518	0,8215	0,8789	0,8863
	YOLO	Rimossi	0,9343	0,9461	0,9435	0,9405
	MRCNN	Presenti	0,9498	0,9214	0,9349	0,9355
	MRCNN	Rimossi	0,9589	0,9470	0,9541	0,9527
	YOLO (Mask)	Presenti	0,9500	0,8208	0,8776	0,8850
	YOLO (Mask)	Rimossi	0,9340	0,9399	0,9402	0,9371
	MRCNN (Mask)	Presenti	0,9518	0,9058	0,9273	0,9287
	MRCNN (Mask)	Rimossi	0,9437	0,9653	0,9570	0,9549

3.3.5 Risultati degli esperimenti cross-dataset

Infine sono stati svolti degli esperimenti cross-dataset, utilizzando il dataset FA nella fase di addestramento e i dataset AirtLab e poi Hockey Fight nella fase di valutazione. Da questi risultati possiamo innanzitutto vedere come le prestazioni dei due sistemi di classificazione utilizzati subiscano un importante crollo, soprattutto nel caso in cui viene usata la rete VGG16. La segmentazione dei fotogrammi porta ad un sensibile abbassamento dell'accuratezza (circa 20%) se si utilizza la rete C3D, quindi la rimozione dello sfondo causa la perdita di feature importanti per lo sviluppo di una buona capacità di generalizzazione da parte del modello. Mentre, la rete VGG16 non raggiunge neanche il 50% di accuracy utilizzando i fotogrammi originali, ma le sue prestazioni migliorano con la rimozione del contesto dai video, pur rimanendo assolutamente insoddisfacenti.

Tabella 3.4: Addestramento con il dataset FA e test sul dataset AirtLab

CNN	**ISM**	**Frame senza detection**	**Spec.**	**Sens.**	**F1-score**	**Accuracy**
C3D			0,2485	**0,9327**	**0,8114**	**0,7085**
	YOLO	Presenti	**0,7972**	0,3725	0,5063	0,5117
	MRCNN	Rimossi	0,6747	0,4594	0,5679	0,5300
VGG16			0,4495	0,6690	0,6256	0,4788
	YOLO	Rimossi	**0,4599**	0,6625	**0,6879**	**0,5960**
	MRCNN	Rimossi	0,2131	**0,8368**	0,5679	0,5300

Tabella 3.5: Addestramento con il dataset FA e test sul dataset Hockey Fight

CNN	**ISM**	**Frame senza detection**	**Spec.**	**Sens.**	**F1-score**	**Accuracy**
C3D			**0,7540**	0,7835	**0,7728**	**0,7688**
	YOLO	Presenti	0,4420	0,9047	0,7359	0,6741
	MRCNN	Rimossi	0,4546	**0,9058**	0,7419	0,6822
VGG16			0,0870	0,8679	0,6907	0,5971
	YOLO	Rimossi	0,3548	**0,9237**	0,7237	0,6429
	MRCNN	Rimossi	**0,3884**	0,9118	**0,7257**	**0,6523**

Capitolo 4

Conclusioni

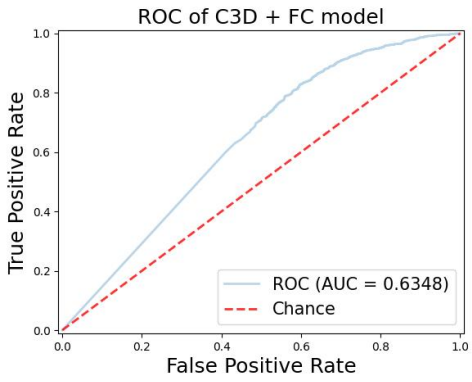
Dai risultati ottenuti possiamo concludere che lo sfondo contiene informazioni utili e ricopre una certa importanza nel processo di apprendimento delle reti neurali, quindi la sua rimozione porta ad un peggioramento dell'accuratezza dei modelli. Questo peggioramento va dal 2% nel dataset Hockey Fight (Tabella 3.2), dove i soggetti e le situazioni rappresentate sono sempre simili, fino a oltre il 10% nel dataset FA (Tabella 3.3), dove invece si ha la maggior variabilità dei contesti all'interno dei video.

Gli esperimenti cross-dataset hanno rivelato, per le reti utilizzate, una grande difficoltà nel generalizzare quanto appreso per applicarlo su un dataset con caratteristiche diverse da quello usato in fase di addestramento. Inoltre, c'è una differenza sostanziale tra il comportamento delle reti C3D e VGG16 in questo caso. Infatti, C3D subisce una grande degradazione delle performance con la rimozione dello sfondo, mentre VGG16 le vede migliorare. Questo ci porta a concludere che la rimozione dello sfondo non è una tecnica in grado di migliorare da sola la robustezza dei sistemi di rilevazione automatica di scene di violenza nei video.

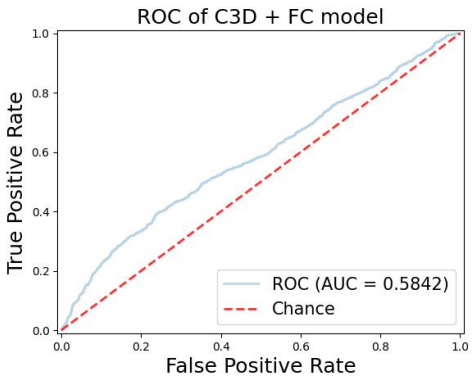
4.0.1 Sviluppi Futuri

Alcuni possibili sviluppi futuri potrebbero essere:

- Sviluppare e valutare altre tecniche al fine di migliorare la capacità di generalizzazione di questi modelli. Ad esempio, si potrebbe aggiungere una fase di Pose Estimation e utilizzare quindi anche le informazioni riguardanti la posa dei soggetti oltre alle feature spazio-temporali;
- Realizzare delle "Saliency maps" sui fotogrammi per indagare in maniera ancor più approfondita quali parti delle immagini influiscono maggiormente sull'output delle reti convoluzionali.

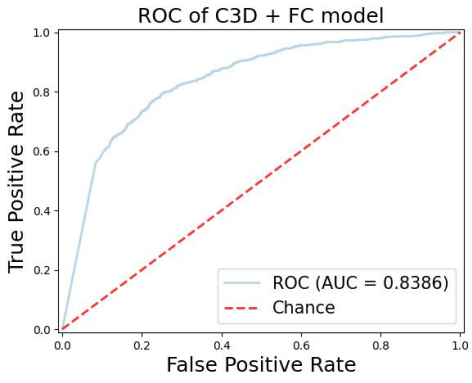


(a) Fotogrammi originali

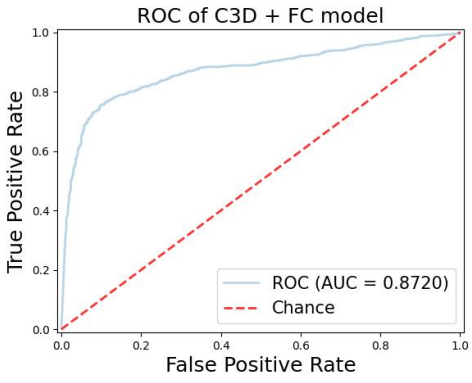


(b) Segmentato con Mask R-CNN con rimozione frame vuoti

Figura 4.1: ROC del modello C3D addestrato su FA e testato su AirtLab

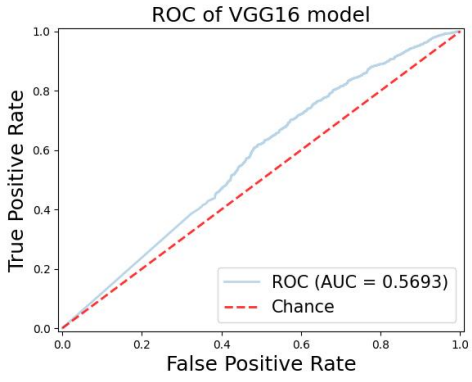


(a) Fotogrammi originali

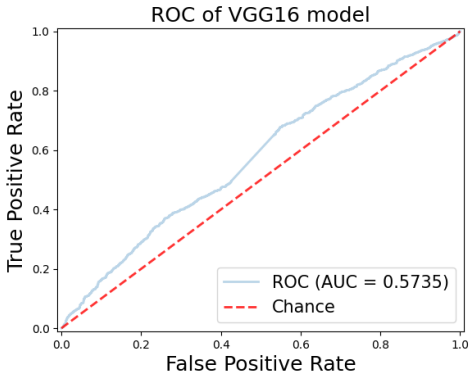


(b) Segmentato con Mask R-CNN con rimozione frame vuoti

Figura 4.2: ROC del modello C3D addestrato su FA e testato su Hockey Fight

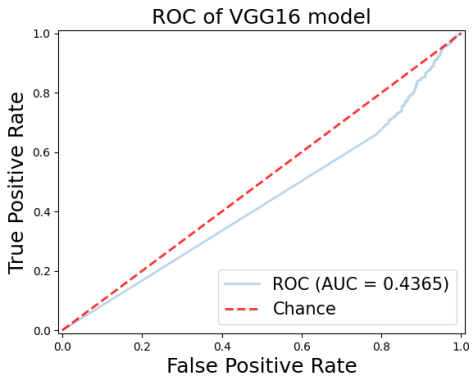


(a) Fotogrammi originali

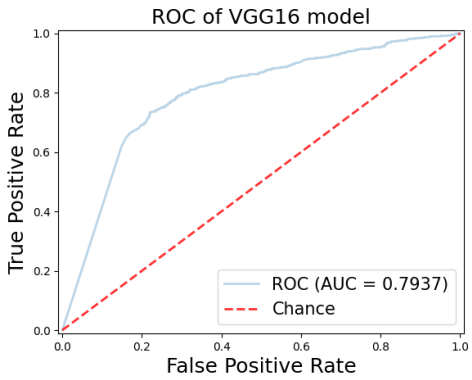


(b) Segmentato con Mask R-CNN con rimozione frame vuoti

Figura 4.3: ROC del modello VGG16 addestrato su FA e testato su AirtLab



(a) Fotogrammi originali



(b) Segmentato con Mask R-CNN con rimozione frame vuoti

Figura 4.4: ROC del modello VGG16 addestrato su FA e testato su Hockey Fight

Bibliografia

[1] [Online], Available: https://blog.csdn.net/jiongnima/article/details/79094159 [Consultato il 05/02/2024].

[2] Newley Purnell Liza Lin. A world with a billion cameras watching you is just around the corner, 2019.

[3] Tahereh Zarrat Ehsan, Manoochehr Nahvi, and Seyed Mehdi Mohtavipour. Learning deep latent space for unsupervised violence detection. *Multimedia Tools and Applications*, 82(8), 2023.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[5] Yanqing Bi, Dong Li, and Yu Luo. Combining keyframes and image classification for violent behavior recognition. *Applied Sciences*, 12(16), 2022.

[6] Mohamed Mostafa Soliman, Mohamed Hussein Kamal, Mina Abd El-Massih Nashed, Youssef Mohamed Mostafa, Bassel Safwat Chawky, and Dina Khattab. Violence recognition from videos using deep learning techniques. In *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 80–85, 2019.

[7] Ming Cheng, Kunjing Cai, and Ming Li. Rwf-2000: An open large scale video database for violence detection. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4183–4190, 2021.

[8] Enrique Bermejo Nievas, Oscar Deniz Suarez, Gloria Bueno García, and Rahul Sukthankar. Violence detection in video using computer vision techniques. In Pedro Real, Daniel Diaz-Pernil, Helena Molina-Abril, Ainhoa Berciano, and Walter Kropatsch, editors, *Computer Analysis of Images and Patterns*, pages 332–339. Springer Berlin Heidelberg, 2011.

[9] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[10] Guillermo Garcia-Cobo and Juan C. SanMiguel. Human skeletons and change detection for efficient violence detection in surveillance videos. *Computer Vision and Image Understanding*, 233:103739, 2023.

[11] D. Paolini. Valutazione e ottimizzazione di reti neurali profonde per il riconoscimento di scene di violenza nei video basate su transfer learning. Tesi di Laurea Triennale, UNIVPM, A.A. 2020/21, Rel. Aldo Franco Dragoni, Corr. Sernani Paolo.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[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] 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].

[15] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[16] 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.

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

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

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[22] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[25] 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.

[26] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos, 2019.

[27] Project jupyter. [Online], Available: https://jupyter.org [Consultato il 05/02/2024].

[28] Google colaboratory. [Online], Available: https://colab.research.google.com [Consultato il 05/02/2024].

[29] Keras. [Online], Available: https://keras.io [Consultato il 05/02/2024].

[30] Opencv. [Online], Available: https://opencv.org [Consultato il 05/02/2024].

[31] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] Scikit-learn. [Online], Available: https://scikit-learn.org/stable [Consultato il 05/02/2024].

[34] 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.