ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

ACTIVITY RECOGNITION E CAPTIONING DA VIDEO CON DEEP LEARNING: ANALISI COMPARATIVA DELLO STATO DELL'ARTE MEDIANTE NUOVI VIDEO SET ARTIFICIALI

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Federico Cichetti

Terza Sessione di Laurea
Anno Accademico 2018 – 2019

PAROLE CHIAVE

Action Recognition
Video Captioning
Deep Learning
Computer Vision
Python

A chiunque mi sia stato vicino,
e mi abbia aiutato a raggiungere questo traguardo.

Introduzione

Il campo della *Computer Vision* (CV) ha assunto una posizione sempre più di rilievo nell'ultimo decennio, con risvolti straordinari nella vita quotidiana delle persone: basti pensare alla moltitudine di applicazioni per dispositivi mobile in grado di analizzare e correggere immagini, applicare filtri tracciando volti e movimenti e riconoscere oggetti complessi all'interno di una fotografia.

Chiaramente, le applicazioni di questo campo vanno ben oltre all'intrattenimento personale: i principali social network adottano algoritmi in grado di riconoscere in automatico *contenuti inadatti*, impianti industriali controllano con apposite telecamere che i pezzi prodotti *non siano difettosi* senza l'intervento umano e i principali *motori di ricerca* propongono immagini simili a quelle ricercate, senza parlare poi dei vari *sistemi di ausilio* per persone con disabilità visive.

L'esplosione di questo campo è solo in parte dovuta all'integrazione pervasiva di foto e videocamere e allo sviluppo tecnologico dei sensori che ne regolano il funzionamento: la maggior parte del merito deve essere associato all'avvento delle *intelligenze artificiali* (AI) e delle tecniche di *deep learning*.

Queste tecniche avanzate sono in grado di *estrarre conoscenza in automatico dai dati*. Poiché una maggior quantità di dati equivale ad una conoscenza più accurata, in un'epoca in cui i contenuti multimediali ci circondano è vertiginosamente aumentato sia l'*interesse* verso questo tipo di problemi che l'*accuratezza* e l'*efficienza* delle soluzioni proposte.

L'analisi di immagini è un problema adatto a *situazioni statiche*, come il riconoscimento di oggetti o l'applicazione di filtri di post-processing. Tuttavia, l'uomo vive in un *mondo dinamico* in cui la dimensione temporale è fortemente presente, ed è quindi limitante avere sistemi che semplicemente ignorano questo aspetto della realtà. Analizzare un'immagine non ci permette di distinguere un uomo in movimento da una figura che semplicemente assume una posa, una macchina che si avvicina da una che si allontana e così via, e l'applicazione di questi sistemi in ambiti in cui il contesto è fondamentale, come quello della *guida autonoma*, può chiaramente portare a risultati *disastrosi*.

Dato che in situazioni reali occorre quindi modellare ciò che viene visto come *sequenze temporali di azioni ed eventi*, lo sviluppo tecnologico e il forte aumento

della quantità di dati disponibili grazie ad Internet hanno reso possibile lo studio dei dati di tipo *video*, i quali contengono a tutti gli effetti anche la dimensione temporale delle immagini che li compongono (detti *fotogrammi* o *frame*).

All'interno di questo elaborato vengono analizzate, comparate e testate alcune delle migliori reti neurali per l'*action recognition* e *captioning* in video, ovvero il problema di *classificare* - nel primo caso - e *descrivere* in linguaggio naturale - nel secondo - le azioni presenti all'interno di un filmato.

Al fine di illustrare quanto il deep learning abbia influito non solo in questo task, ma in tutto l'ambito della Computer Vision, vengono presentati i principali *componenti* di queste reti e si offre una panoramica sul loro funzionamento.

Verranno illustrate, inoltre, le differenze tra i principali dataset attualmente disponibili per questi problemi e si presenterà un nuovo *dataset artificiale*, con il quale si possono generare una quantità arbitraria di video su cui addestrare modelli di riconoscimento attività in un'ambientazione simil-industriale.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 1** - Analisi preliminare del problema e dei dati disponibili;
- **Capitolo 2** - Panoramica sulle varie tecnologie presenti in letteratura utilizzabili per il problema posto;
- **Capitolo 3** - Analisi dettagliata del problema, delle migliori soluzioni esistenti, del loro funzionamento e delle loro differenze chiave;
- **Capitolo 4** - Analisi dei risultati dei test effettuati sulle architetture e i dataset selezionati e presentazione del nostro nuovo dataset artificiale;
- Conclusioni e direzioni future.

Indice

1 Analisi del problema **1**
1.1 Contesto applicativo 1
1.2 Dataset disponibili 5
1.2.1 Dataset di Action Recognition 5
1.2.2 I benefici del pre-training 7
1.2.3 Dataset per il video captioning 8
1.3 Obiettivi 9

2 Le tecnologie disponibili **11**
2.1 Introduzione al Machine Learning 11
2.1.1 Apprendimento supervisionato 12
2.1.2 Apprendimento non supervisionato 13
2.1.3 Apprendimento per rinforzo 14
2.1.4 Discesa del gradiente 15
2.1.5 Validazione del modello 17
2.2 Deep Learning 18
2.2.1 Struttura 19
2.2.2 Reti Neurali Convoluzionali (CNN) 20
2.2.3 Architetture CNN state-of-the-art 24
2.2.4 Reti Neurali Ricorrenti (RNN) 28
2.2.5 Paradigma encoder-decoder 31
2.2.6 Meccanismi di attention 32
2.3 Immagini e video come dati di input 34
2.3.1 Architettura two-stream 35
2.3.2 Architettura a convoluzioni 3D 36
2.3.3 Altre architetture 37

3 Analisi delle soluzioni **39**
3.1 Video Captioning 39
3.1.1 Image Captioning 40
3.1.2 Analisi del problema 41
3.1.3 Presentazione dei modelli state-of-the-art 43

3.2 Nota sulle soluzioni di Video Captioning 48
3.3 Action Recognition 49
3.3.1 Valutazione di sistemi di classificazione 49
3.3.2 Analisi del problema 51
3.3.3 Presentazione dei modelli state-of-the-art 53
3.4 Osservazioni finali 60

4 Risultati degli esperimenti 63
4.1 Valutazione dei modelli 63
4.1.1 Tecnologie utilizzate 63
4.1.2 Analisi dei risultati ottenuti 66
4.2 Costruzione di un dataset artificiale 69
4.2.1 Ambientazione 69
4.2.2 Struttura di un video 70
4.2.3 Implementazione 73
4.2.4 Analisi dei risultati 78

Conclusioni e sviluppi futuri 83

Ringraziamenti 85

Bibliografia 87

Elenco delle figure

1.1 Rappresentazione di un'immagine digitalizzata 2
1.2 Esempio di immagini da due dataset che mostra quanto la complessità dei problemi di *Image Recognition* sia aumentata nel tempo. 3
1.3 Tabella riassuntiva dei dataset disponibili per il problema del video captioning. 8

2.1 Differenze tra le due tecniche di programmazione. 12
2.2 Rappresentazione grafica del problema del clustering. 14
2.3 Modello di base nell'apprendimento di rinforzo. 14
2.4 Rappresentazione grafica della discesa del gradiente. La dimensione x è da intendersi come l'iperspazio di ricerca dei parametri, mentre su y si ha il valore della funzione di costo. Il punto di minimo in cui $\theta = \hat{\theta}$ è il punto da raggiungere in cui i parametri hanno il loro valore ideale. 16
2.5 Rappresentazione del problema dell'overfitting 17
2.6 Diversi livelli di astrazione estratti dalle reti neurali. 18
2.7 Struttura tipica di una rete neurale. 19
2.8 Struttura di un neurone artificiale. 20
2.9 Collegamenti tra i *livelli* di una rete convoluzionale 21
2.10 Esempio di un'operazione di convoluzione 22
2.11 Esempio di feature map dati due filtri su un'immagine in input. 22
2.12 Rappresentazione di una rete convoluzionale in cui sono ben evidenziate le multiple feature map prodotte ad ogni livello . . . 23
2.13 Rete LeNet-5 24
2.14 La prima versione del modulo Inception. 26
2.15 L'architettura VGGNet, versione VGG16. 27
2.16 Un *residual block* di ResNet. 27
2.17 Funzionamento della backpropagation in ResNet. 28
2.18 Una semplice RNN formata da un unico neurone che propaga il suo output. 29
2.19 Schema di una cella LSTM, alternativa alla cella tradizionale. . 30

2.20 Esempio di un'architettura encoder-decoder formata da due RNN separate. 32
2.21 Meccanismo di attention su un modello di traduzione dall'inglese al francese. 33
2.22 Architettura utilizzata in [1] che mostra l'applicazione del paradigma encoder-decoder e della spatial attention. 34
2.23 Architettura two-stream per l'action recognition in un video. . . 36
2.24 Una convoluzione 3D. A sinistra, la rappresentazione tridimensionale del video (ogni quadratino è un pixel e ogni "strato" un fotogramma), al centro il kernel della convoluzione e a destra la feature map prodotta. 37

3.1 Rappresentazione di una classica soluzione per l'image captioning. 40
3.2 Esempio di una soluzione completa per il video captioning . . . 43
3.3 Architettura del framework HACA 44
3.4 Architettura del framework MGSA 45
3.5 Architettura di un modulo GARU (Gated Attention Recurrent Unit) 46
3.6 Architettura del modello M&M TGM 47
3.7 Una matrice di confusione per un sistema di classificazione a due classi (*positive* e *negative*). 50
3.8 I macrogruppi di architetture per l'analisi video. 52
3.9 Funzione softmax. 52
3.10 Un semplice classificatore di immagini con un layer di *softmax* per produrre le probabilità di appartenenza alle classi. 53
3.11 L'architettura Hidden Two-Stream 54
3.12 Architettura Inflated Inception-V1. 56
3.13 Rappresentazione del framework TSN nelle sue fasi: segmentazione del video, sampling degli snippets, estrazione di features, aggregazione dei risultati e predizione delle probabilità. 57
3.14 Architettura SlowFast e i suoi due percorsi. 59

4.1 Differenze tra l'utilizzo di virtual machine (a destra) e container Docker (a sinistra) per l'esecuzione di più applicazioni. 65
4.2 Le due versioni dell'ambiente modellato per lo svolgimento delle attività riprese nei video. 70
4.3 Alcuni fotogrammi che mostrano elementi randomici all'interno dei video del dataset. 72
4.4 Matrici di confusione ottenute testando i modelli sulla versione di blender-industrial con 1210 video (370 video di validation). . 80

Capitolo 1

Analisi del problema

In questo capitolo viene contestualizzato il lavoro svolto, vengono definiti gli obiettivi del progetto ed analizzati i dataset disponibili, in modo da fornire una visione più specifica del problema.

1.1 Contesto applicativo

Il campo della *computer vision* affascina da svariati anni il mondo dell'informatica, grazie agli enormi sviluppi e alle numerose applicazioni in progetti realmente utili per l'uomo e la società.

I computer hanno diversi modi per essere consapevoli dell'ambiente che li circonda. Ad esempio, nel mondo dei sistemi embedded si usano tantissimi tipi diversi di sensori per raccogliere e processare informazioni provenienti dall'esterno. GPS, giroscopi e accelerometri integrati anche nei nostri telefoni cellulari permettono di studiare la nostra posizione e come ci stiamo muovendo, sonar e sensori di prossimità riescono a notificare la presenza di ostacoli o brusche variazioni di calore.

L'essere umano non è dotato di sensori così precisi, ma dalla sua parte ha lo strumento della *vista*, che gli permette di vedere ostacoli anche a grande distanza, avere riferimenti per orientarsi e riconoscere cosa sta succedendo davanti a lui. Questa capacità innata di comprendere una situazione quasi istantaneamente è un grande vantaggio sulle macchine che, al contrario, devono essere programmate per individuare complessi *pattern* nei dati grezzi ed eterogenei che raccolgono dai vari sensori.

Dare la possibilità ai computer di *vedere* in modo analogo agli esseri viventi è stato in questo senso un enorme sviluppo tecnologico che ha permesso la costruzione di sistemi impensabili fino a una decina di anni fa: auto capaci di guidarsi da sole, motori di traduzione in grado di riconoscere frasi scritte su un cartello o a mano attraverso l'input di una fotocamera e così via.

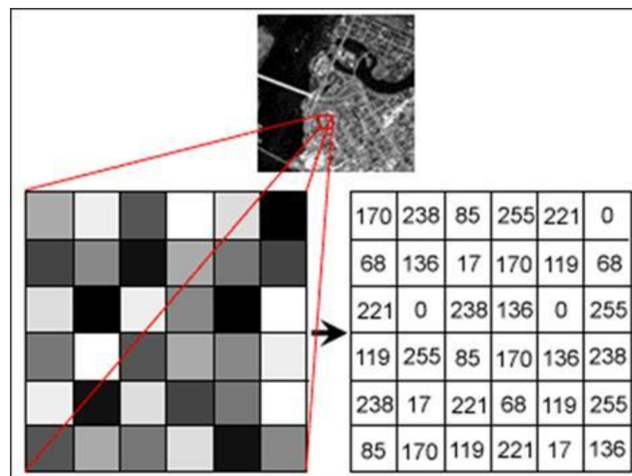


Figura 1.1: Rappresentazione di un'immagine digitalizzata
Fonte: https://www.researchgate.net/figure/Digital-image-representation-by-pixels-vii_fig2_311806469

Tutte queste applicazioni si basano sul *riconoscimento del contenuto di immagini.* Per un computer, un'immagine è rappresentabile come una *matrice di pixel* (Figura 1.1), semplici valori numerici che indicano un'intensità rispetto a una scala predefinita.

Ad esempio, in un'immagine grayscale, un pixel assumerà un valore tra 0 (nero) e 255 (bianco) che ne quantifica la luminosità. Le immagini a colori funzionano concettualmente nello stesso modo, ma per memorizzare le informazioni aggiuntive si fa uso di più *canali.* Un'immagine RGB, ad esempio, ha tre canali (Red, Green e Blue) e un singolo pixel avrà un valore per ognuno di questi corrispondente all'intensità con cui il colore del canale partecipa alla *sintesi additiva* per formare il colore finale.

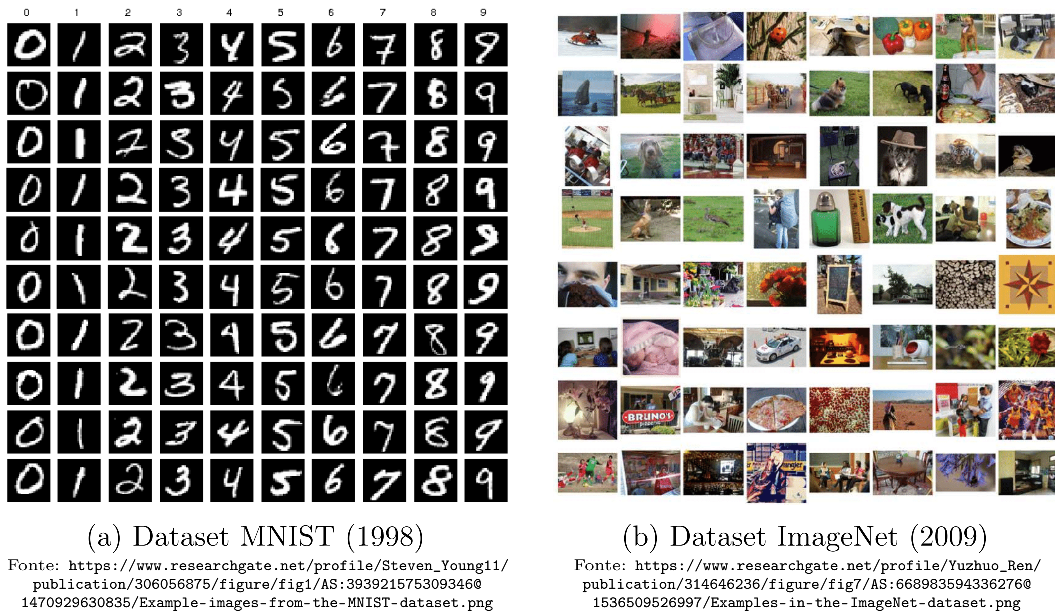
Gli oggetti all'interno di un'immagine possono essere identificati trovando dei *pattern* tra i pixel, cioè schemi di valori che si ripetono in modo riconoscibile.

Questo task è molto complicato per diversi motivi:

- *Alta dimensionalità* dei dati di input. Un singolo pixel di un'immagine a colori secondo lo standard RGB può essere rappresentato in uno *spazio tridimensionale* (una dimensione per ogni canale, con dominio tra 0 e 255). Tuttavia, le immagini possono raggiungere risoluzioni abbastanza alte. Lo standard Full HD per i video, ad esempio, denota una risoluzione di 1920x1080 pixel, con la quale si supera il milione di dimensioni.

- *Alta variabilità* dei pattern. Uno stesso soggetto può essere fotografato da diverse angolazioni, comparire in parti diverse dell'immagine, ed essere esposto ad illuminazioni differenti, ma il sistema dovrebbe comunque essere in grado di riconoscerlo in modo univoco.

Risulta quindi molto difficile - se non impossibile - programmare *manualmente* sistemi di riconoscimento immagini, a meno che i domini non siano forzatamente ridotti ed il task molto specifico.



(a) Dataset MNIST (1998)

Fonte: `https://www.researchgate.net/profile/Steven_Young11/publication/306056875/figure/fig1/AS:393921575309346@1470929630835/Example-images-from-the-MNIST-dataset.png`

(b) Dataset ImageNet (2009)

Fonte: `https://www.researchgate.net/profile/Yuzhuo_Ren/publication/314646236/figure/fig7/AS:668983594336276@1536509526997/Examples-in-the-ImageNet-dataset.png`

Figura 1.2: Esempio di immagini da due dataset che mostra quanto la complessità dei problemi di *Image Recognition* sia aumentata nel tempo.

Il dataset MNIST (Figura 1.2a) disponibile presso `http://yann.lecun.com/exdb/mnist/`, è universalmente considerato l'*hello world* di questo tipo di problemi. Si tratta di un dataset di immagini 28x28 in bianco e nero che raffigurano delle cifre scritte a mano da diverse persone. Il task è chiaramente quello di identificare le cifre disegnate e classificare, tramite questo, le immagini.

Fashion MNIST (Figura 1.2b) è un problema analogo in cui immagini nello stesso formato rappresentano diversi capi di abbigliamento che devono essere associate alla loro *classe di appartenenza* (T-shirt, scarpe, ...)

Nonostante la (relativamente) bassa dimensionalità e la specificità del task (ci sono solo 10 classi in entrambi i dataset), è molto difficile affrontare il problema cercando di costruire manualmente algoritmi per individuare i pattern necessari. Usando il *machine learning*, al contrario, è possibile addestrare un algoritmo al riconoscimento dei pattern in maniera *automatica* attraverso i dati: si tratta quindi dell'alternativa più precisa e veloce. L'automatismo per il quale l'algoritmo viene addestrato a riconoscere i pattern è detto *training* e verrà trattato più avanti in questa sede.

I migliori risultati su entrambi i dataset sono stati ottenuti tramite *reti neurali*, meccanismi di machine learning complessi, ma anche molto potenti ed efficaci, che saranno spiegati nel prossimo capitolo.

Le reti neurali sono state impiegate con successo anche quando, col tempo, i dataset si sono fatti più grandi e articolati. Attualmente, è ImageNet [2] il dataset con cui le soluzioni di machine learning relative all'analisi di immagini si confrontano maggiormente. ImageNet (Figura 1.2b) ha la qualità di essere molto vasto e vario, il che è un bene perché gli algoritmi possono imparare a riconoscere molti più pattern e per ognuno hanno più esempi per affinare i loro parametri.

Negli ultimi anni, con l'esplosione dei social network e dei siti dedicati ai video, è fiorito l'interesse per questo complesso tipo di dato. Avere sistemi che riconoscono cosa sta succedendo all'interno di un video può avere svariate applicazioni, come ad esempio:

- Motori di ricerca video in grado di cercare con precisione azioni complesse o a cui si possono descrivere in maniera anche abbastanza dettagliata i video target;
- Applicazioni in grado di descrivere in linguaggio naturale video o interi film a persone con gravi disabilità visive;
- Automazione della rilevazione di attività svolte dagli operai di un'industria su alcuni macchinari (manutenzione e altri lavori).

I metodi che sono stati sviluppati per analizzare i video hanno come punto di partenza comune gli studi fatti sull'Image Recognition, poiché i video possono a tutti gli effetti essere considerati sequenze di immagini. La ricerca di pattern nelle immagini diventa, però, nei video una ricerca *tridimensionale*, perché per comprendere pienamente un'*attività* deve essere analizzata anche la dimensione temporale in cui si svolge.

Per esempio, se il video in input mostra una persona che interagisce con una porta, da una singola immagine si può comprendere il soggetto e il contesto in cui si trova, ma non se si tratta del movimento di *apertura* o quello di *chiusura*: risulta quindi fondamentale analizzare le *dipendenze* che attraversano i fotogrammi *nel tempo* in cui è possibile trovare pattern proprio come all'interno della dimensione spaziale delle immagini.

La rappresentazione di queste dipendenze temporali è uno dei motivi per cui il problema del riconoscimento di azioni all'interno dei video è così complesso (e, di conseguenza, così studiato). Ci sono diverse linee di pensiero e architetture che nel corso degli anni sono state proposte per rappresentare al meglio questa *terza dimensione*, e si analizzeranno le principali nel corso dei prossimi capitoli.

1.2 Dataset disponibili

In letteratura, il problema del riconoscimento del contenuto di un video è detto:

- *Video Captioning* se lo scopo finale è quello di ottenere una *descrizione* in linguaggio naturale del video;
- *Action Recognition* se invece l'obiettivo è quello di *classificare* un video, ovvero determinare quali tra una lista di azioni predefinite sono quelle svolte nella scena.

A prescindere dal problema in analisi, un'altra distinzione importante riguarda la lunghezza dei video:

- I video *trimmed* sono video che mostrano lo svolgimento di una singola attività. Solitamente si tratta di brevi clip (entro i 30 secondi) probabilmente ottenute tagliando video più lunghi, da cui il nome;
- I video *untrimmed* hanno, invece, una durata maggiore (fino a 15 minuti nel dataset AVA [3]) e possono mostrare più scene, persone ed azioni.

Ognuna delle quattro possibili combinazioni di questi parametri denota un problema diverso e, di conseguenza, una lista di dataset disponibili diversa.

1.2.1 Dataset di Action Recognition

C'è una grande varietà di dataset sull'Action Recognition. I video possono provenire da diverse fonti: alcuni vengono collezionati da YouTube, altri da riprese di incontri sportivi e altri ancora, come nel caso del dataset *Charades*, sono stati commissionati via *crowdsourcing* a diverse persone.

I dataset più importanti a livello storico sono UCF101 [4] (evoluzione dei più vecchi UCF11 e UCF50) e HMDB51 [5]. La grandissima maggioranza dei modelli è testata prima di tutto su questi due dataset per capirne l'efficacia generale. I video sono molto brevi (entro i 10 secondi), il che rende i dataset relativamente leggeri ed utilizzabili agevolmente.

Le classi assegnate ai video di UCF101 descrivono l'azione in modo molto generale e si concentrano principalmente sulle attività che le persone svolgono ad alto livello, ad esempio saltare la corda, fare degli squat, suonare uno strumento, tirare con l'arco...

In HMDB51 alcune classi riguardano espressioni e movimenti facciali delle persone coinvolte nei video (ridere, parlare, masticare, baciare, ...), ma si trovano anche molte azioni di movimento generale (alzarsi, girarsi, calciare una

Datasets per l'Action Recognition				
Tipologia	Nome	Anno	Azioni	Clip
trimmed	HMDB51	2011	51	6,766
	UCF101	2012	101	13,320
	Sports-1M	2014	487	1,133,158
	Charades	2016	157	9,848
	Kinetics 400	2017	400	306,245
	Something-Something v1	2017	174	108,499
	Kinetics 600	2018	600	495,547
	Something-Something v2	2018	174	220,847
	Moments in Time	2018	339	900,964
	Kinetics 700	2019	700	650,317
untrimmed	ActivityNet v1.2	2015	100	13,213
	THUMOS15	2015	101	18,404[1]
	ActivityNet v1.3	2016	203	28,108
	AVA	2017	80	430[2]

Tabella 1.1: Tabella riassuntiva dei principali dataset disponibili per l'action recognition.

palla, ...). Molto spesso, i risultati di classificazione su questo dataset sono scarsi, con il modello correntemente state-of-the-art che raggiunge l'82,48% di precisione contro il 98% su UCF101 (fonte: `https://paperswithcode.com/task/action-recognition-in-videos`). Questo è dovuto al fatto che, nonostante le poche classi, molte si somigliano (sword, sword exercise, fencing e draw sword, per esempio, sono facilmente confondibili). Sono inoltre presenti pochi video di training, il che abbassa di molto la qualità delle predizioni.

Il dataset Kinetics 400 [6], con le sue successive estensioni, viene spesso riconosciuto come un diretto successore di UCF101 e HMDB51, in quanto mantiene l'idea di fondo di avere tante clip brevi (10 secondi, in questo caso) con una singola azione generale da riconoscere. Ciò che cambia è il numero di video e di classi che Kinetics è in grado di offrire, anche grazie all'aumento della varietà dei video disponibili in rete e motivato dalla crescita della difficoltà dei problemi riguardanti il tipo di dato video e la conseguente *complessità* dei modelli.

[1] Il training set è UCF101.

[2] Clip tratte da film lunghe circa 15 minuti annotate temporalmente.

Charades [7] contiene video più lunghi in cui vengono svolte diverse azioni in sequenza in un ambiente domestico. Gli autori notano che costruendo dataset di video utilizzando piattaforme online come YouTube viene a formarsi un notevole *bias* verso azioni fuori dal comune, come quelle svolte in eventi sportivi o per intrattenimento, poiché gli utenti caricano online video che ritengono possano avere un certo *interesse*. Azioni quotidiane e *monotone* come "alzarsi dal letto" o "lavare i piatti" non sono quindi rappresentate in maniera abbastanza *genuina* in questi dataset. In questo senso, Charades è stato costruito affidando al crowd-sourcing la produzione di video in cui si rappresentano azioni quotidiane o "*noiose*" che sono però anche più *realistiche*. Oltre alle classi di azioni e alle indicazioni temporali di quando esse avvengono, il dataset fornisce per ogni video anche una descrizione in linguaggio naturale: perciò può essere affrontato anche anche con metodi di video captioning.

Il dataset Something-Something [8], infine, è in realtà pensato per un problema ibrido tra l'action recognition e il video captioning. Le 174 classi disponibili sono *template* per frasi in linguaggio naturale, ovvero strutture predeterminate con, però, degli spazi vuoti da completare per creare frasi ("*Dropping <something> next to <something>*"). Le azioni sono molto specifiche e alcune classi richiedono una precisa analisi spaziale: ci sono ad esempio le classi "*Dropping <something> into <something>*", "*Dropping <something> in front of <something>*" e "*Dropping <something> onto <something>*" che si differenziano dalla prima in base a *dove* cade il primo oggetto rispetto al secondo.

1.2.2 I benefici del pre-training

La generalità delle classi di UCF101, HMDB51 e Kinetics li rende poco consoni al riconoscimento di attività complesse: molto spesso, infatti, si usano semplicemente per permettere un confronto standard tra un nuovo modello e quelli precedenti.

Un utilizzo più interessante che se ne fa è invece quello del *pre-training*. La fase di *training* di un modello (che verrà ripresa nei dettagli nei prossimi capitoli) è un processo *iterativo* in cui parte dei video del dataset è passata, uno alla volta, come input al modello, il quale *si migliora* iterazione dopo iterazione grazie ai nuovi dati. Dato che il compito di un modello è essenzialmente quello di *ritrovare nei nuovi dati i pattern imparati precedentemente*, le prime iterazioni tendono a rendere molto *instabile* il modello, perché partendo da zero, cioè senza aver *mai* visto un video prima, non c'è ancora abbastanza conoscenza su questi pattern per determinarli.

Per fare un esempio, un modello che vede per la prima volta una persona aprire una porta può assumere che la scelta della classe "apertura porta" sia determinata da fattori irrilevanti, come il colore del muro o i vestiti che indossa

la persona. Se alla successiva iterazione riceve un video in cui l'apertura della porta avviene in una stanza diversa e con diversi attori, sarà costretto ad aggiustare il tiro, ponendo più attenzione alle parti in comune nei due video. Solo dopo moltissime iterazioni il modello sarà *stabile*, cioè avrà imparato una rappresentazione dell'apertura di una porta abbastanza *generale* da non dover cambiare totalmente con ogni nuovo video in input.

Quella del *pre-training* è un'idea semplice ma efficace che pone rimedio a questo problema: invece di addestrare da zero un modello su un task complesso, lo si addestra prima su un altro task *simile*, in modo che il training inizi con già un po' di vantaggio. Questa tecnica è anche detta *transfer learning*.

Un grande beneficio del pre-training deriva dal fatto che, avendo già una solida base di conoscenza incorporata, il modello avrà bisogno di *meno iterazioni* per raggiungere i risultati migliori, e conseguentemente *meno dati*. Questo è molto importante in un ambito come quello dello studio dei video, dove è molto costoso, sia in termini di tempo che di spazio, creare dataset di dimensioni appropriate.

1.2.3 Dataset per il video captioning

Dataset	Domain	No of Videos	No of Clips	Duration (hrs)	No of Sent.	Avg Word	Vocab. Size	Temp. Anno.
MSVD [Chen and Dolan, 2011]	Open	1,970	1,970	5.3	70,028	8.7	13,010	✓
TACoS [Regneri *et al.*, 2013]	Cooking	127	3,290	10.1	18,818	9.0	1,413	✓
YouCook [Das *et al.*, 2013]	Cooking	88	-	2.3	3,502	12.6	2,329	×
TACoS-multilevel [Rohrbach *et al.*, 2014]	Cooking	185	14,105	15.7	52,593	8.3	2,864	×
MPII-MD [Rohrbach *et al.*, 2015]	Movie	94	68,337	73.6	68,375	9.6	24,549	✓
M-VAD [Torabi *et al.*, 2015]	Movie	92	48,986	84.6	55,904	9.3	18,269	✓
MSR-VTT [Xu *et al.*, 2016]	Open	7,180	10,000	41.2	200,000	9.3	29,316	×
TGIF [Li *et al.*, 2016]	Open	100,000	-	86.1	125,781	10.6	11,806	×
ActivityNet Captions [Krishna *et al.*, 2017]	Open	20,000	73,000	849.0	73,000	13.5	10,646	✓
DiDeMo [Hendricks *et al.*, 2017]	Open	10,464	26,892	144.2	41,206	7.5	7,587	✓

Figura 1.3: Tabella riassuntiva dei dataset disponibili per il problema del video captioning.

Fonte: [9]

I dataset di video captioning associano ad ogni video una o più frasi in linguaggio naturale con lo scopo di addestrare i modelli a produrre frasi simili al riconoscimento di tali azioni.

Nella figura 1.3 sono elencati i principali dataset per questo problema. La colonna Avg Word indica il numero di parole medio per le frasi presenti nel dataset, mentre *Temp. Anno.* indica se sono presenti anche annotazioni temporali per le descrizioni che indicano a quale punto del video si riferiscono.

Da quando è stato creato nel 2016, la maggior parte delle soluzioni di video captioning sono addestrate sul dataset *MSR-VTT* (Microsoft Research - Video To Text) [10] che è il più ricco e vario, offrendo 10.000 clip video con 20 annotazioni umane per clip in 20 categorie diverse. Altri dataset abbastanza noti sono MSVD, un dataset di video di carattere generale molto più piccolo di MSR-VTT, TGIF che comprende oltre 100.000 GIF selezionate dal sito Tumblr, YouCook e TACoS con video di cucina e M-VAD, MPII-MD con clip tratte da film.

1.3 Obiettivi

L'obiettivo principale di questo lavoro di tesi è quello di riportare i risultati più importanti che sono stati raggiunti in questo campo, analizzando nel dettaglio le soluzioni allo stato dell'arte e testando personalmente alcune di quelle disponibili pubblicamente. L'idea di fondo è quella di capire quale di queste architetture sarebbe maggiormente in grado di riconoscere *attività lavorative in ambito industriale* in modo da automatizzarne l'inserimento in uno storico.

I dataset disponibili pubblicamente sono o troppo generali o troppo legati ad altri campi perché i risultati siano ottimali per il nostro problema, ma a livello ipotetico rimane possibile studiare e testare i modelli sui dataset disponibili e cercare di capire quale soluzione potrebbe funzionare meglio avendo la giusta quantità di dati.

Dopo aver dato una panoramica sull'utilizzo delle intelligenze artificiali nel campo della Computer Vision, si analizzeranno prima le soluzioni di *Video Captioning* e successivamente si sposterà l'attenzione sul problema dell'*Action Recognition*.

Infine, per far fronte alla mancanza di un dataset specifico per il nostro problema, verrà costruito e presentato un nuovo *dataset artificiale* con cui sarà possibile generare un numero arbitrario di video pensati per emulare possibili attività in ambito industriale con un certo grado di realismo. I video sono ambientati in uno spazio tridimensionale virtuale creato utilizzando il software Blender.

Si confronterà il comportamento dei modelli allo stato dell'arte anche su tale dataset per trarre le conclusioni finali.

Capitolo 2

Le tecnologie disponibili

In questa prima parte verranno descritte più nello specifico le tecnologie alla base delle complesse architetture analizzate ed impiegate nei capitoli successivi.

2.1 Introduzione al Machine Learning

Il Machine Learning viene spesso descritto, citando Tom M. Mitchell, come una disciplina scientifica che "studia programmi ed algoritmi informatici in grado di *migliorarsi in automatico attraverso l'esperienza*", o in maniera più formale:

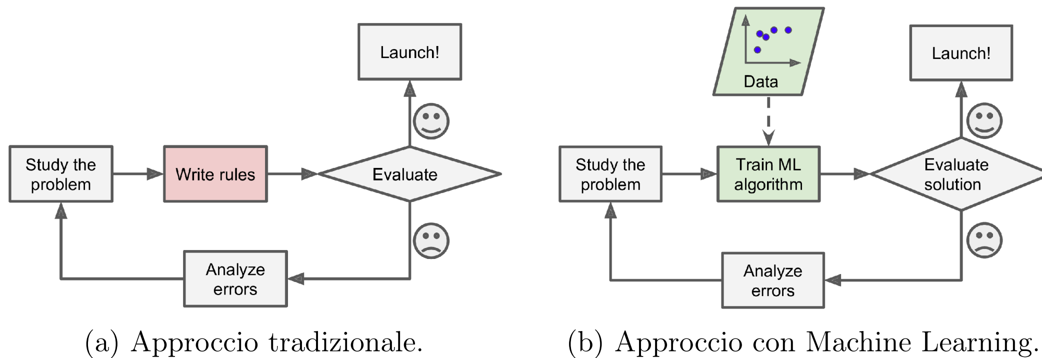
> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E".* [11]

Si tratta quindi di un diverso approccio alla programmazione di computer per lo svolgimento di un determinato compito: se *tradizionalmente* lo sviluppo di un programma richiede una *conoscenza esperta* del task per cui viene costruito, in modo da poter tradurre l'*algoritmo decisionale* applicato nel mondo reale in codice, le tecniche di Machine Learning fanno in modo che tale algoritmo sia *estratto* direttamente dai dati che si hanno a disposizione (Figura 2.1).

L'algoritmo costruito in questa maniera viene chiamato *modello*, in quanto si tratta essenzialmente di un'*approssimazione matematica* con moltissimi *parametri* che *modella* la realtà descritta dai dati il più fedelmente possibile.

Questo nuovo approccio è utile per diversi motivi:

- Innanzitutto, consente di *velocizzare* enormemente la scrittura dei programmi per i quali l'algoritmo decisionale è complesso;



(a) Approccio tradizionale. (b) Approccio con Machine Learning.

Figura 2.1: Differenze tra le due tecniche di programmazione.
Fonte: [12]

- In secondo luogo, permette di *migliorare* gli stessi algoritmi decisionali applicati tradizionalmente, scegliendone automaticamente altri potenzialmente migliori;
- Infine, le tecniche di Machine Learning sono abbastanza flessibili da gestire l'adattamento a *nuovi dati* in caso di cambi repentini di regole.

Al giorno d'oggi, esempi di applicazioni in cui viene utilizzato il Machine Learning sono davvero ovunque: dagli articoli consigliati che possiamo trovare entrando su un qualsiasi sito di e-commerce, alle previsioni del tempo, dalle chatbot agli assistenti vocali che riconoscono la nostra voce.

Come è già stato introdotto nello scorso capitolo, il processo iterativo con cui il modello viene estratto da un dataset è detto *addestramento* o *training*. In base al problema in considerazione e alla struttura dei dati a disposizione, l'apprendimento del modello può essere classificato in tre modi diversi, ovvero:

- Apprendimento supervisionato;
- Apprendimento non supervisionato;
- Apprendimento per rinforzo.

2.1.1 Apprendimento supervisionato

L'apprendimento *supervisionato* è il tipo di training più semplice e diffuso. In parole povere, richiede che ogni dato passato all'algoritmo di learning sia comprensivo dell'indicazione della *soluzione attesa*.

Un tipico task affrontato con l'apprendimento supervisionato è quello della *classificazione*, ovvero quello di raggruppare i dati assegnandovi *label* in base ad alcune loro caratteristiche.

Il dataset MNIST già citato è pensato per questo tipo di problemi, poiché lo scopo finale è quello di classificare un'immagine in base alla cifra che mostra. Il dataset fornisce per ogni immagine anche il corretto label, quindi è facile per il modello *verificare* se una predizione si è rivelata corretta o meno.

L'addestramento per i modelli di classificazione è un processo condotto *iterativamente* su tutti i dati del dataset, riproponendoli più volte fino a che il modello non *impara* a riconoscere gli aspetti davvero importanti per effettuare una predizione corretta. Il processo è il seguente:

- Inserimento di un dato come input per l'algoritmo;
- Confronto delle predizioni dell'algoritmo con gli output attesi noti a priori;
- Aggiustamento dei parametri in maniera proporzionale al confronto condotto nel punto precedente.

Verrà affrontata nella sezione 2.1.4 l'ultima fase di questo processo, ovvero il modo in cui il modello riesce a *migliorare in automatico* le proprie predizioni.

I dataset attraverso cui i modelli imparano hanno spesso dimensioni troppo grandi per poter essere inseriti interamente in memoria, specialmente per quanto riguarda immagini e video. Per questa ragione, è comune effettuarne una suddivisione in *batch*, blocchi che contengono solo una parte dei dati (per i dataset di video trimmed, molto spesso i batch sono da 16 o 32 video a seconda delle disponibilità hardware).

Per poter capire se l'apprendimento sta funzionando o meno, il dataset viene inoltre passato al modello più e più volte: ogni iterazione di questo processo viene detta *epoch*. Il numero di epoch è un valore importante, perché se eccessivamente alto rende molto lungo il training, mentre se troppo basso causa un modello *poco rappresentativo* dei dati su cui si basa. Un numero corretto, cioè tale da raggiungere un errore medio abbastanza basso da poter considerare il modello valido, può essere trovato solo *euristicamente.*

2.1.2 Apprendimento non supervisionato

Il problema dell'apprendimento supervisionato è che prima di passarli ad un modello, i dati devono essere analizzati manualmente da un *supervisore*, il quale ha il compito di indicare quali siano i label reali utilizzando l'algoritmo tradizionale.

A volte potrebbe essere difficile o dispendioso etichettare i dati, come nei casi di problemi in cui possono esserci molte sfaccettature e casi limite da tenere in considerazione.

Uno di questi problemi è quello del *clustering* (Figura 2.2), in cui si richiede di individuare dei gruppi all'interno di una popolazione di dati tali che i componenti dei vari gruppi abbiano caratteristiche il più simili possibili.

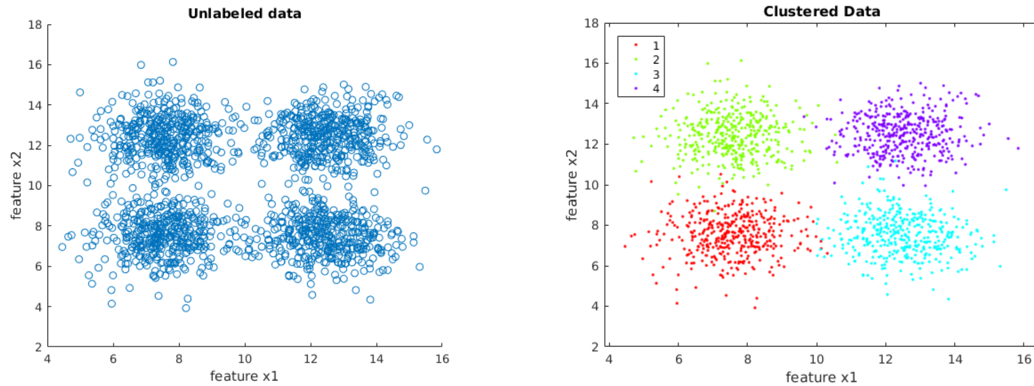


Figura 2.2: Rappresentazione grafica del problema del clustering.
Fonte: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/unsupervised_learning.html

Il task è l'analogo della classificazione nell'apprendimento supervisionato, ma invece di definire a priori le possibili classi di appartenenza dei dati, i dataset di clustering contengono *dati sparsi* all'interno dei quali l'algoritmo deve *autonomamente* trovare dei pattern.

Il learning non supervisionato è più complesso di quello con supervisione esterna, ma altrettanto utile poiché permette di individuare collegamenti nascosti in grandi moli di dati.

2.1.3 Apprendimento per rinforzo

L'apprendimento per *rinforzo*, o *reinforcement learning*, rappresentato in figura 2.3, è molto diverso dai metodi presentati finora. L'algoritmo viene chiamato *agente*, poiché assume un ruolo *attivo* sull'*ambiente* che lo circonda con cui può interagire captandone variazioni e scegliendo le azioni da compiere in risposta.

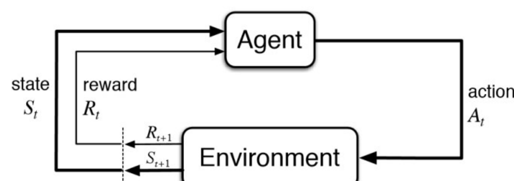


Figura 2.3: Modello di base nell'apprendimento di rinforzo.
Fonte: https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg

Il learning, in questo caso, funziona con un *sistema a punteggio*: l'agente ottiene un *premio* se esegue azioni che lo avvicinano al completamento del task, mentre riceve delle *punizioni* se ciò che sceglie di fare è distruttivo o controproducente. Più che l'effettivo completamento del task, l'enfasi è posta sulla *strategia* o *policy* che l'agente deve imparare a seguire per ottenere i punteggi migliori.

Il reinforcement learning è un campo più di nicchia rispetto ai tipi di apprendimento classico, anche se recentemente si è parlato molto di AlphaGo (`https://deepmind.com/research/case-studies/alphago-the-story-so-far`), un progetto di DeepMind che nel 2017 ha battuto il campione mondiale ad una partita di Go applicando la policy di vittoria imparata dopo aver analizzato milioni di partite. Recentemente sono stati introdotti ambienti virtuali, come la Gym di OpenAI (`https://gym.openai.com/`) in cui è possibile addestrare agenti osservandone i progressi in tempo reale. Già negli esempi basilari di quest'ultimo tool è possibile trovare algoritmi che giocano a videogiochi analizzandone le schermate o robot che imparano gradualmente a camminare o spostare oggetti nello spazio.

2.1.4 Discesa del gradiente

Il metodo della *discesa del gradiente* è l'algoritmo che sta al cuore di tutto il machine learning. Matematicamente, si tratta di un algoritmo di *ottimizzazione* usato per trovare i *parametri* tali per cui il valore di una *funzione di costo* sia minimo.

L'algoritmo è *iterativo*, perciò tali parametri non vengono calcolati istantaneamente ma, partendo da un punto random nell'*iperspazio* di scelta, ci si sposta gradualmente verso un altro punto e poi un altro ancora, fino a raggiungere quello in cui la funzione di costo ha il valore minimo.

Nel machine learning si usa questa tecnica per trovare i parametri decisionali migliori per un modello. Per fare ciò, la funzione di costo viene costruita in modo che combaci con la *funzione di errore* del modello, ovvero una funzione che calcola quanto il valore predetto dal modello per i dati di training si discosta da quello reale.

Solitamente si sceglie come funzione di errore la *MSE* (*Mean Squared Error*), ovvero la *media dei quadrati degli errori* sulle predizioni, ma esistono molte alternative. In ogni caso, nel punto di minimo della funzione scelta si ha il modello più *accurato*, poiché i valori predetti e quelli reali saranno quelli più simili.

Graficamente, è possibile rappresentare una discesa del gradiente come in figura 2.4.

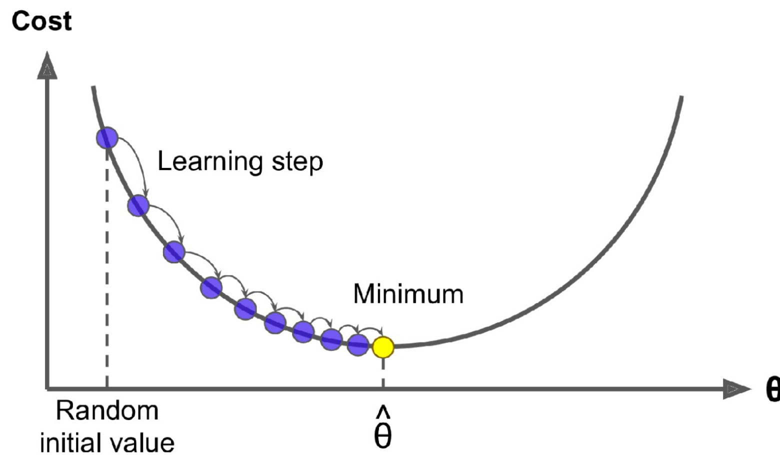


Figura 2.4: Rappresentazione grafica della discesa del gradiente. La dimensione x è da intendersi come l'iperspazio di ricerca dei parametri, mentre su y si ha il valore della funzione di costo. Il punto di minimo in cui $\theta = \hat{\theta}$ è il punto da raggiungere in cui i parametri hanno il loro valore ideale.

Fonte: [12]

A livello pratico, gli step effettivi che seguono l'inizializzazione random dei parametri del modello sono i seguenti:

- Generazione delle predizioni per i dati del dataset;

- Individuazione del punto attualmente in considerazione. Questo equivale a calcolare la funzione di errore in base alle predizioni effettuate;

- Calcolo del *gradiente* della funzione nel punto individuato. Il gradiente di una funzione a più variabili è il vettore delle *derivate parziali* per ognuno dei parametri: in parole povere, si tratta dei valori della *pendenza* in ogni dimensione in un determinato punto. Per un esempio più pratico, si può immaginare di avere una pallina sul bordo della funzione di errore e di calcolare il gradiente per il punto in cui si trova. In questo modo, si scopre quale *direzione* è la più *ripida* in quel punto, quindi verso dove la pallina rotolerà per raggiungere il fondo, cioè il punto di *minimo*;

- Attuazione del passo di *discesa* del gradiente: si sottrae al punto attuale un vettore proporzionale al gradiente, il che equivale a modificare i parametri del modello in modo che si abbiano le variazioni maggiori su quelli più lontani dai loro valori ideali. Nell'analogia fatta in precedenza, questo è il momento in cui si fa rotolare, per qualche istante, la pallina verso il fondo;

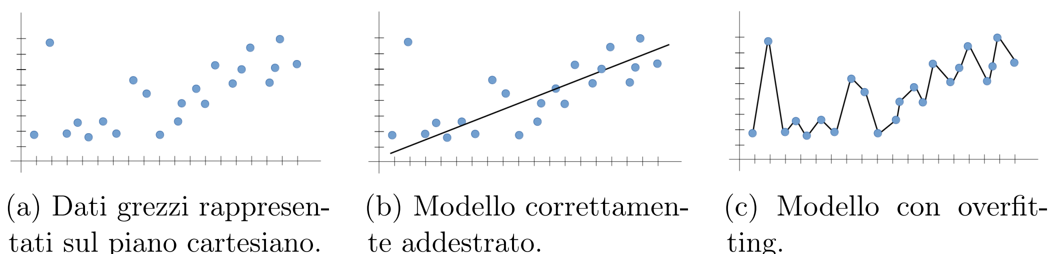
- Ricominciare da capo finché non viene raggiunto il *punto obiettivo*. In questo punto, il modello sarà *stabile* perché l'errore e conseguentemente il gradiente e le variazioni dei parametri saranno minimi.

2.1.5 Validazione del modello

Uno dei problemi maggiori nel machine learning è quello dell'*overfitting*. L'incremento del numero di parametri permette infatti di modellare i dati in maniera più *esatta*, ma, oltre all'aumento del *costo computazionale* dell'apprendimento, se il modello è troppo complesso per il problema in considerazione, si rischia che l'approssimazione della realtà effettuata da esso sia *eccessivamente* legata ai dati di addestramento.

Nella maggior parte dei casi, un algoritmo di machine learning deve essere in grado di reagire a *nuove situazioni* non presenti nei dataset: non si tratta cioè di modellare una funzione per rispondere con *assoluta certezza* all'intero spettro di possibili casi, ma di creare un modello abbastanza *generale* da poter gestire con *buone approssimazioni* il maggior numero di eventualità possibili, comprese quelle mai viste.

Se il modello perde la generalità con cui dovrebbe affrontare il problema, si dice che va in *overfitting*. La figura 2.5 dimostra graficamente questo problema.



(a) Dati grezzi rappresentati sul piano cartesiano.

(b) Modello correttamente addestrato.

(c) Modello con overfitting.

Figura 2.5: Rappresentazione del problema dell'overfitting
Fonte: https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690

Per capire se un modello è andato in overfitting, è necessario applicare una tecnica chiamata *hold-out* che consiste nel suddividere in due parti non sovrapposte il dataset di addestramento:

- Il *training set*, che contiene la maggior parte dei dati (di solito circa il 70%) ed è quello su cui viene effettivamente addestrato il modello;

- Il *validation set*, che viene tenuto da parte durante l'addestramento così da contenere dati mai visti su cui viene verificata l'accuratezza del modello. In particolare:

- Se i risultati ottenuti nelle predizioni sono molto buoni solo sul training set ma non sul validation, allora c'è overfitting;
- Se su entrambi gli insiemi il modello si comporta *circa* allo stesso modo, allora si può dire che si tratta di un'approssimazione abbastanza generale e quindi valida.

2.2 Deep Learning

Il *Deep Learning* è una nuova area del Machine Learning basata sull'impiego di strutture dette *reti neurali* ispirate al funzionamento del cervello umano per la risoluzione di problemi complessi.

Le aree in cui il Deep Learning è usato maggiormente sono quelle in cui i dati *non sono strutturati* o possono assumere diverse forme e dimensioni, come ad esempio il Natural Language Processing (NLP) o l'analisi di immagini. In particolare, i successi ottenuti in quest'ultimo campo, come ad esempio il tasso d'errore dello *0,17%* sul dataset MNIST [13], sono stati il *punto di partenza* per tutte le soluzioni moderne di video analysis.

L'unità fondamentale di una rete neurale è il *neurone artificiale*, un modello matematico che emula il funzionamento di un neurone biologico. L'intera struttura è una fitta *rete* di questi neuroni che *comunicano* tra loro mandando *segnali* di output in risposta a segnali di input ricevuti a loro volta da altri neuroni.

L'addestramento di una rete neurale è una questione complessa, ma nella pratica ciò che viene fatto è semplicemente cambiare il modo in cui i neuroni rispondono ai segnali di input fino ad ottenere un basso tasso di errore: nella rete, anche una sola risposta diversa può provocare una reazione a catena tra i neuroni e globalmente causare un miglioramento del sistema.

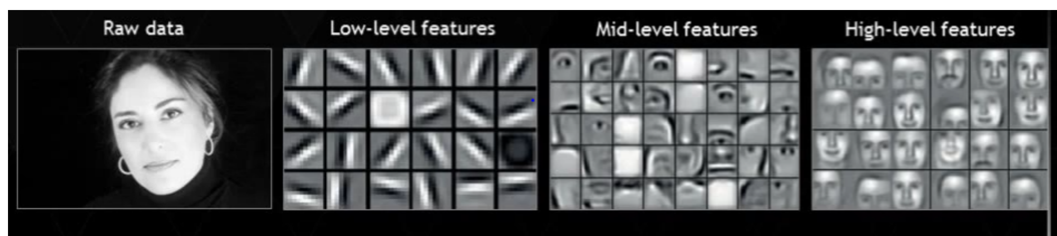


Figura 2.6: Diversi livelli di astrazione estratti dalle reti neurali.
Fonte: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

I neuroni sono disposti su più *strati* (*layer*), i quali rappresentano anche l'*astrazione* con cui viene trattato il problema. Nei layer più bassi, si impara a riconoscere *pattern semplici* (forme geometriche e associazioni tra colori, nel

caso dello studio di immagini), mentre nei layer superiori queste conoscenze di base sono mescolate e riunite a creare una prospettiva di più *alto livello*, come si può vedere in figura 2.6.

2.2.1 Struttura

La *stratificazione* è il vero punto di forza delle reti neurali. I layer rendono le architetture molto più complesse degli algoritmi classici per il machine learning, ma questa complessità si traduce in una maggiore efficacia.

Di solito si descrive una rete neurale attraverso i livelli di cui si compone e i loro collegamenti. Il livello di partenza viene chiamato *input layer* e molto spesso non fa alcun tipo di elaborazione sull'input, ma semplicemente lo *propaga* ai livelli successivi sotto forma di segnale.

I layer intermedi sono detti *hidden layers* e, come già detto, rappresentano i vari gradi di astrazione con cui l'input viene studiato.

Infine, l'*output layer* deve avere tanti neuroni quanti sono i valori attesi in output. Per esempio, se si sta costruendo una rete per il dataset MNIST e si vogliono ottenere le *probabilità* dell'immagine in input di far parte di ognuna delle 10 classi, servirà un output layer con 10 neuroni. La figura 2.7 mostra un esempio per una piccola rete neurale.

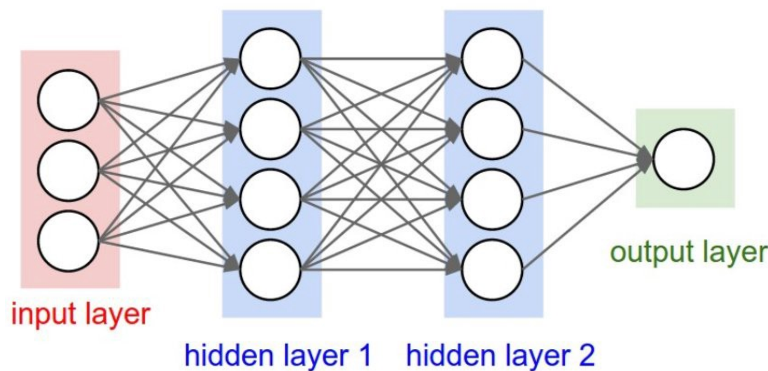


Figura 2.7: Struttura tipica di una rete neurale.
Fonte: `http://cs231n.github.io/neural-networks-1/`

Se ogni neurone di un livello è connesso a tutti quelli del livello successivo (come in figura 2.7) si parla di una *rete densa*. Le reti dense sono le architetture neurali più semplici, ma ovviamente non sono le uniche: alcuni neuroni possono essere direttamente collegati ad altri che si trovano diversi livelli più avanti, o anche allo stesso neurone che all'elaborazione temporalmente successiva si vede ricevere in input tutto o parte del suo output precedente. Diverse configurazioni sono possibili, e la ricerca ne produce continuamente di nuove e migliori.

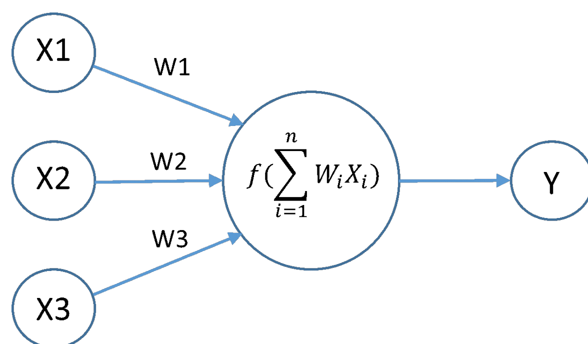


Figura 2.8: Struttura di un neurone artificiale.

Ogni singolo neurone (Figura 2.8), indipendentemente dalla configurazione della rete, riceve in input un insieme di valori all'interno di un vettore x (detto, in certi casi, *tensore*) ed elabora internamente il valore y, generalmente compreso tra 0 e 1, da restituire in output. L'elaborazione comporta il calcolo di una media pesata degli input attraverso i pesi W e l'inserimento del valore ottenuto all'interno di una *funzione di attivazione* f che produce l'output.

Arrivati in fondo alla rete, si confronta la predizione con l'output atteso, si calcola l'errore con la discesa del gradiente (o altri metodi più performanti) e viene effettuato il passo detto di *backpropagation*: si scorre, cioè, la rete all'indietro e si cambiano i pesi dei neuroni che più hanno contribuito alla produzione dell'errore. I pesi vengono quindi continuamente aggiustati finché la rete non impara dei buoni parametri e si stabilizza.

In reti molto profonde possono accadere problemi di *sparizione* o *esplosione del gradiente* (*vanishing/exploding gradients*): in pratica, man mano che l'errore viene fatto scendere nella rete nel passo di backpropagation, può capitare che i gradienti rimpiccioliscano sempre di più fino a provocare variazioni di parametri praticamente nulle ai livelli più bassi, o che al contrario il loro valore diventi vertiginosamente alto facendo oscillare i pesi durante tutto il training e ritardandone la convergenza. Questi due problemi sono tradizionalmente affrontati con meccanismi di *normalizzazione* della rete, ma in alcuni casi, come quando avviene nelle *reti ricorrenti*, è necessario fare uso di tecniche più specifiche, come verrà mostrato più avanti.

2.2.2 Reti Neurali Convoluzionali (CNN)

Le reti neurali *convoluzionali* sono spesso preferite alle reti dense nei problemi di Computer Vision, dato che raggiungono un'ottima accuratezza riducendo l'elevatissimo numero di neuroni e parametri richiesti in queste ultime.

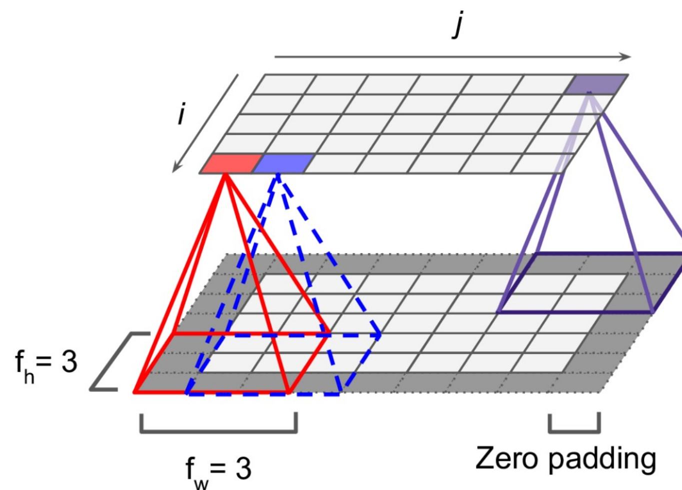


Figura 2.9: Collegamenti tra i *livelli* di una rete convoluzionale
Fonte: [12]

Il meccanismo su cui si basano le CNN è ispirato al reale funzionamento della *corteccia visiva* umana: i neuroni hanno un *campo recettivo* più piccolo dell'intero campo visivo, perciò ognuno di essi si occupa solo di una piccola regione dell'immagine ricevuta. I campi recettivi di due neuroni possono parzialmente sovrapporsi, ma l'importante è che alla fine venga coperta l'intera immagine di input.

I livelli di una rete convoluzionale fanno esattamente questo: i neuroni di un livello non sono connessi a tutti quelli del livello precedente, ma solamente a quelli in un certo *range*. Pensando ai livelli della rete come *matrici* impilate una sopra l'altra, si può immaginare che ogni neurone, cioè cella di una matrice, riceva come input i valori in output delle celle comprese all'interno di una piccola area nella matrice sottostante (figura 2.9). Il campo recettivo dei neuroni più esterni potrebbe comprendere anche porzioni al di fuori del livello precedente, ma questo è facilmente evitabile allargando la matrice e circondandola con degli zeri (*zero-padding*, come in figura) in modo che tali input non siano considerati, oppure restringendola, cioè non considerando le righe e colonne più esterne.

I pesi dati agli output del livello precedente possono essere rappresentati come piccole immagini della dimensione del campo recettivo del neurone dette *filtri* o *kernel*. Ogni "pixel" all'interno di questi filtri contiene il valore del peso da associare ad un determinato neurone di input del livello sottostante.

Dato che i campi recettivi sono parzialmente sovrapposti, l'output di un neurone al livello $t-1$ può essere pesato diversamente dai neuroni del livello t che lo prendono in considerazione.

Quindi, al livello t ogni neurone produrrà la media pesata di un'area dell'immagine al livello $t-1$ secondo la griglia di pesi determinata dal suo

filtro. Quest'operazione di *media pesata locale* è molto comune nel campo dell'elaborazione di immagini ed è chiamata *convoluzione* (Figura 2.10).

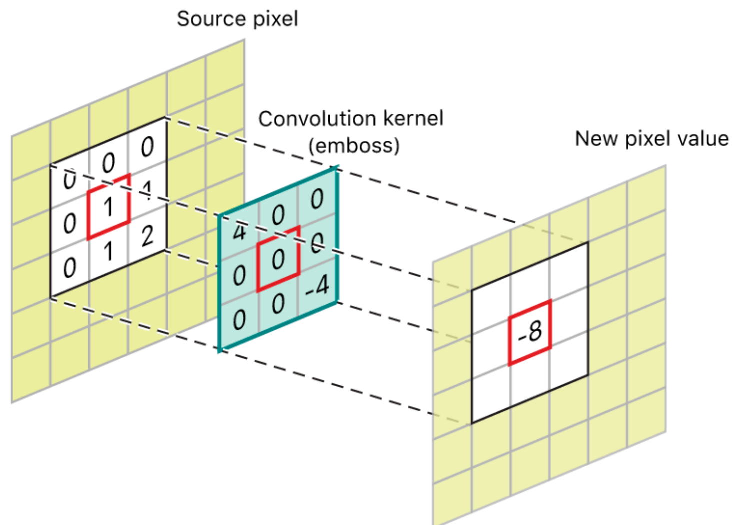


Figura 2.10: Esempio di un'operazione di convoluzione

Un livello in cui tutti i neuroni utilizzano lo stesso filtro produce un'immagine che mette in evidenza le aree di quella ricevuta in input più attivate da suddetto filtro: tale immagine è detta *feature map* (Figura 2.11).

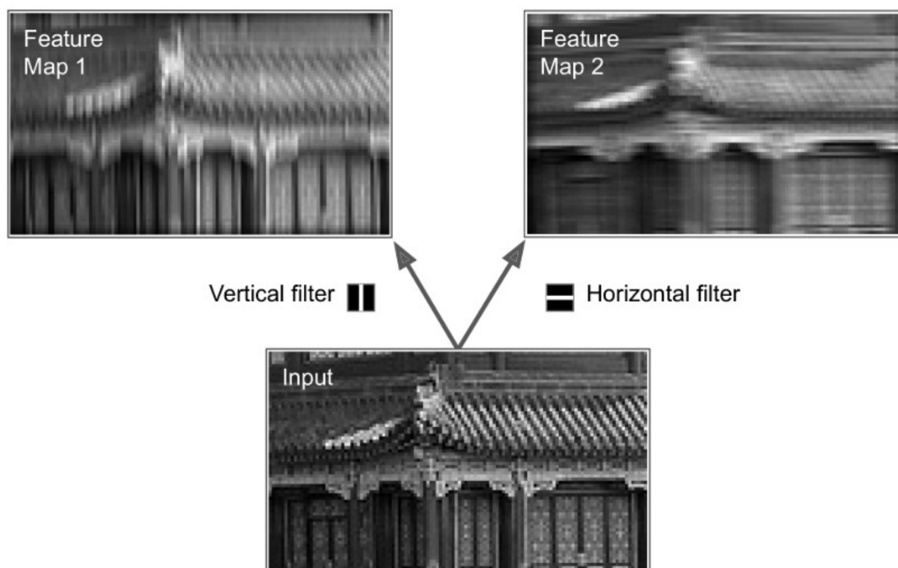


Figura 2.11: Esempio di feature map dati due filtri su un'immagine in input.
Fonte: [12]

Un livello può però produrre più di una feature map: basta usare tanti filtri diversi. Avere più feature map sullo stesso livello permette di mettere in risalto un maggior numero di forme e pattern in contemporanea.

La rappresentazione più accurata di una CNN è quindi quella di vettori tridimensionali impilati, dove la profondità di ogni livello è il numero di feature maps prodotte (Figura 2.12). Un neurone di una specifica feature map è connesso a tutti i neuroni nel suo campo recettivo in tutte le feature maps del livello precedente.

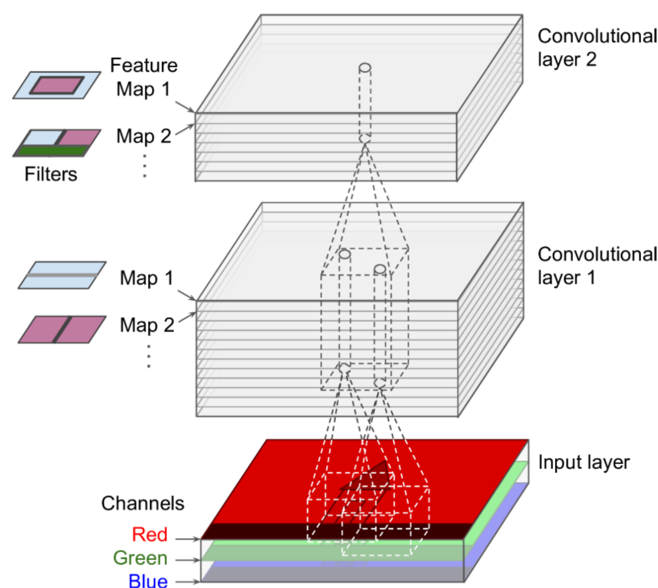


Figura 2.12: Rappresentazione di una rete convoluzionale in cui sono ben evidenziate le multiple feature map prodotte ad ogni livello
Fonte: [12]

Oltre ai livelli convoluzionali, le CNN possono essere dotate anche di livelli di *pooling*. Il loro scopo è quello di fare un *subsampling spaziale* delle feature map, cioè di ridurne le dimensioni, in modo da diminuire il carico computazionale.

I neuroni dei livelli di pooling si comportano come quelli dei livelli convoluzionali, ma i loro filtri sono detti *funzioni di aggregazione*. Le due funzioni di aggregazione più usate sono quella di media, e in tal caso si parla di *Average Pooling*, e quella di massimo (*Max Pooling*). Nel primo caso, il valore finale di un neurone di pooling è la *media* dei valori dei neuroni nel suo campo recettivo, mentre nel secondo caso è molto semplicemente il *valore massimo* tra questi. Spesso la Max Pooling è preferita perché più veloce da computare.

Un altro lato positivo dei livelli di pooling è quello di introdurre *invarianza* nel modello: infatti, riducendo la risoluzione dell'immagine vengono annullate

piccole transizioni, rotazioni e variazioni di scala dei soggetti e il modello diventa meno sensibile a spostamenti di singoli pixel.

Spesso, le CNN alternano livelli convoluzionali a livelli di pooling in modo da rendere l'immagine sempre meno risoluta ma più profonda: infatti ai livelli bassi è importante imparare filtri semplici per estrarre le feature di basso livello, ma salendo verso l'alto le feature map devono essere aggregate per ottenere filtri sempre più complessi ed iniziano a diventare poco rilevanti i dettagli primitivi.

2.2.3 Architetture CNN state-of-the-art

Con i blocchi basilari appena visti, è possibile costruire architetture molto complesse in grado di ottenere risultati straordinari nei problemi reali di image e video analysis.

Quelle che hanno ottenuto le accuratezze migliori sono spesso rese disponibili online, con licenza open-source, preaddestrate su dataset enormi come ImageNet e pronte per essere riutilizzate in task diversi secondo il principio del pre-training (sezione 1.2.2).

Dato che queste reti vengono spesso riutilizzate persino dalle soluzioni di video analysis, nelle sottosezioni qui di seguito si presentano le principali.

LeNet-5

L'architettura LeNet-5 [14] (Figura 2.13) è stata proposta da Yann LeCun nel 1998 ed è considerata la base di tutte le soluzioni moderne. Si tratta essenzialmente dell'architettura già introdotta nella sezione precedente, in cui layer di convoluzioni che estraggono le feature map dall'immagine sono inframezzati con layer di pooling che ne riducono le dimensioni. Essendo MNIST il dataset in esame, questi layer sono seguiti da una piccola rete densa che produce i risultati della classificazione per le dieci classi di cifre.

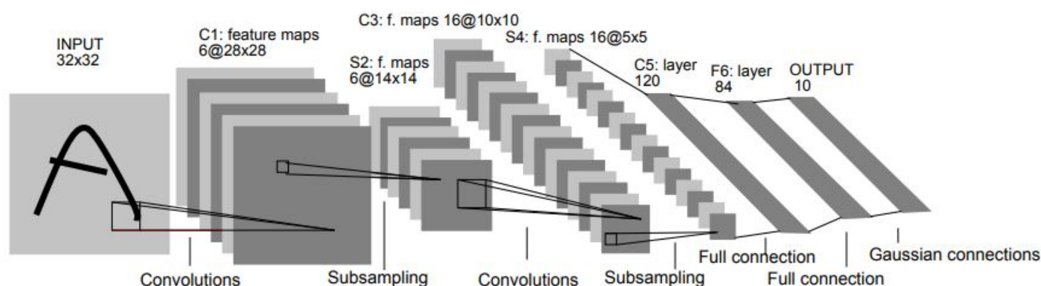


Figura 2.13: Rete LeNet-5

Fonte: [14]

GoogLeNet e Inception

Creata nel 2014, la prima versione di Inception Net, detta anche GoogLeNet [15], ha rivoluzionato fortemente la costruzione di architetture per CNN.

Nel periodo che separa LeNet-5 da GoogLeNet c'è stato un grande sviluppo in questo campo, anche grazie ai dataset di immagini sempre più grandi e numerosi e alle migliori risorse di calcolo. Tuttavia, molte delle architetture di questo periodo si limitavano ad allungare le reti, cambiare dimensioni dei kernel e, in generale, variare gli *iperparametri* di architetture simili a LeNet senza realmente introdurre novità strutturali.

Inception Net si discosta da questo trend, affrontando diversi problemi logistici riguardanti l'*efficienza* da un lato e la necessità di creare reti *molto profonde* per avere una migliore separazione dei livelli di astrazione dall'altro.

All'aumentare della profondità di una rete aumentano anche i *parametri dell'algoritmo*, il che causa un forte rallentamento nel training con i possibili problemi di vanishing/exploding gradients di cui si è trattato in precedenza. Va anche considerato il fatto che una rete con moltissimi parametri tende ad andare facilmente in *overfitting*.

Un ulteriore problema nella costruzione di una CNN è quello della scelta delle *dimensioni dei kernel* per ogni layer. Filtri piccoli servono per estrarre pattern *locali*, mentre i filtri più grandi riconoscono pattern ad una granularità più grossa: quando però uno stesso soggetto può essere molto piccolo, oppure in primo piano all'interno dell'immagine, è difficile fare una scelta generale che sia adatta ad entrambi i casi.

GoogLeNet introduce il *modulo Inception* (Figura 2.14), una serie di layer convoluzionali speciali in cui vengono applicati in contemporanea tre filtri di *dimensioni diverse* sull'input (5x5, 3x3, 1x1). I risultati delle convoluzioni vengono poi concatenati prima di essere passati al layer successivo: in questo modo l'output ha una profondità elevata e l'analisi avviene con *granularità* diverse, preservando sia le informazioni locali che quelle globali.

Come si può vedere dalla figura 2.14, le operazioni di convoluzione sono precedute da convoluzioni con filtri 1x1. In pratica, se l'input di un modulo Inception ha dimensione (F, H, W), dove F è la profondità, cioè il numero di filtri del layer precedente, e H e W sono larghezza e altezza dell'immagine, eseguire una convoluzione con F_1 filtri convoluzionali 1x1 su di esso produce un output con dimensioni (F_1, H, W). Se F_1 è minore di F, viene ridotta la profondità dei dati in input e quindi il numero di parametri necessario: in questo modo la rete viene *alleggerita* ed è più *efficiente*.

Il modello finale è molto profondo. La parte centrale è composta da 9 moduli Inception di fila e l'intera architettura può vantare ben *27 layer*.

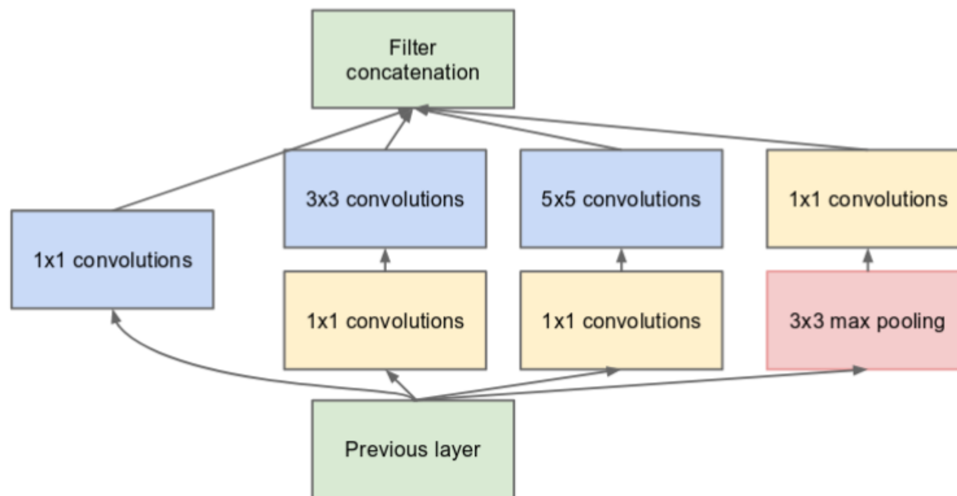


Figura 2.14: La prima versione del modulo Inception.
Fonte: [15]

Ulteriori studi su questa architettura hanno portato alla rielaborazione del modulo Inception e quindi a diverse versioni della rete, attualmente indicate con v1, v2, v3 e v4.

- Inception v2 [16] è uno studio sull'efficienza delle convoluzioni proposte. Introduce tre varianti del modulo Inception in cui i layer convoluzionali vengono riorganizzati in modo più intelligente, ad esempio sostituendo i filtri 5x5 con due convoluzioni 3x3 in successione, o rendendo i filtri *n*x*n* combinazioni di 1x*n* e *n*x1;
- Inception v3 introduce piccoli cambiamenti tecnici per ridurre l'overfitting;
- Inception v4 modifica la parte iniziale della rete e introduce i *reduction block* per ridurre anche le dimensioni W e H dell'immagine.

Esistono anche due versioni di *Inception-ResNet*, architetture ibride nate a seguito del successo di ResNet, una rete analizzata più avanti.

VGGNet

VGGNet [17] (in figura 2.15) ha un'architettura classica, con la tipica alternanza tra convoluzioni e layer di pooling. La vera innovazione sta nella sostituzione di kernel molto grandi con successioni di kernel più piccoli ed efficienti, come in Inception v2. Questo, come già detto, riduce il numero di parametri e permette alla rete di essere più profonda e di effettuare un'analisi più ricca dell'input.

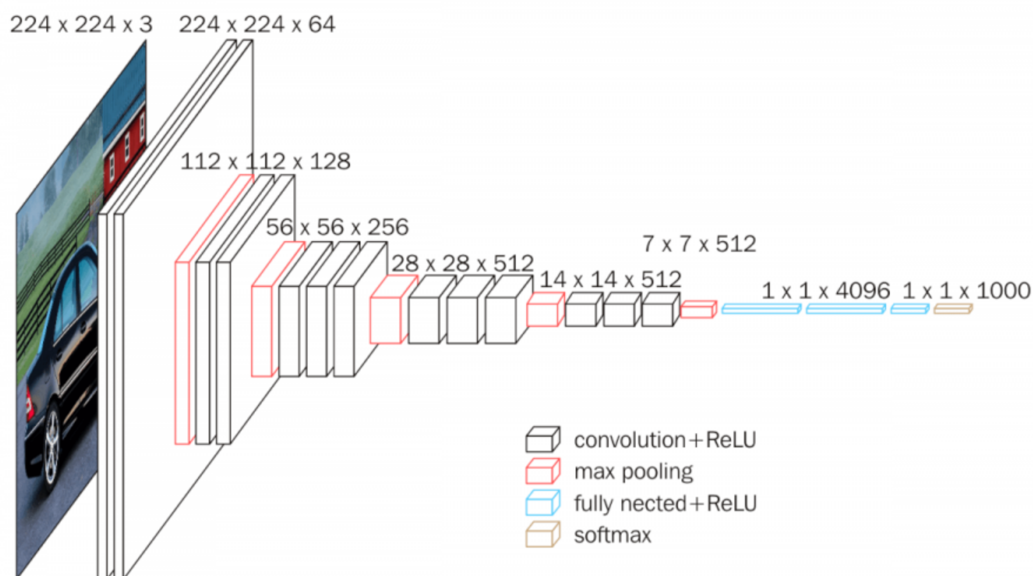


Figura 2.15: L'architettura VGGNet, versione VGG16.
Fonte: `https://neurohive.io/en/popular-networks/vgg16/`

ResNet

Il principio di allungare la *profondità* della rete piuttosto che la sua larghezza è giustificato dal fatto che, con pochi layer, il modello rischia di andare facilmente in overfitting. Inoltre, per problemi complessi, un layer molto grande può richiedere tantissimi parametri. Avere una rete profonda con layer semplici migliora la *generalità* e l'*efficienza* della rete, perciò lo sviluppo delle architetture è sempre stato legato ad un'espansione in questa dimensione.

ResNet [18], con i suoi 152 layer (ne esistono varianti da 34, 50 e 101) è l'esempio che più rappresenta questo trend. Per mantenere stabile una rete così profonda, gli autori fanno uso delle *skip connection*, ovvero di scorciatoie che permettono all'output dei neuroni di *saltare* alcuni dei layer successivi. ResNet ha una struttura *modulare*, in cui ogni blocco, detto *residual block*, è una piccola rete neurale con una skip connection (figura 2.16).

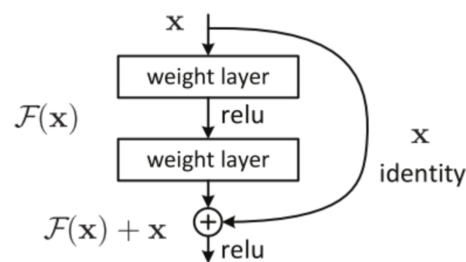


Figura 2.16: Un *residual block* di ResNet.
Fonte: `https://neurohive.io/en/popular-networks/resnet/`

L'uso dei residual blocks ha il vantaggio di velocizzare il training poiché, inizialmente, verrà modellata una funzione simile alla *funzione identità* piuttosto

che una funzione casuale, dato che gli input possono raggiungere i livelli più alti senza essere processati. Le skip connections permettono inoltre ai layer di più alto livello di iniziare ad apprendere quando ancora i layer dei primi livelli stanno stabilizzando i propri pesi.

Un altro vantaggio è l'abbattimento dei problemi di vanishing gradients: infatti, i gradienti nel passo di backpropagation hanno sempre una via *libera* per potersi propagare nel resto della rete, anche quando un residual block ne blocca il progresso, come mostra la figura 2.17.

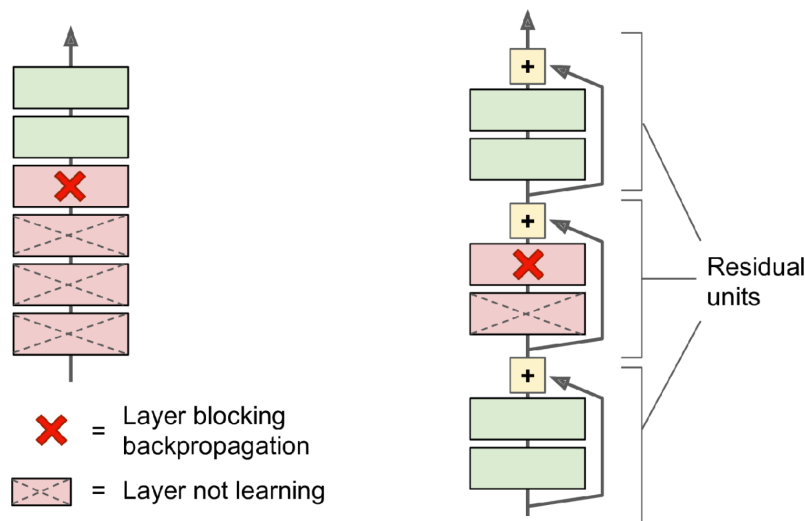


Figura 2.17: Funzionamento della backpropagation in ResNet.
Fonte: [12]

2.2.4 Reti Neurali Ricorrenti (RNN)

Le reti neurali convoluzionali sono ottimizzate per task basati su immagini, come l'*image recognition*. Chiaramente, le architetture che analizzano video non possono essere formate semplicemente da una CNN, poiché un video è una *sequenza* di fotogrammi.

Una strategia naïf per affrontare il problema potrebbe essere quella di produrre una predizione per ogni singolo fotogramma o per un sottoinsieme di essi per poi aggregarle in qualche modo ottenendone una finale, ad esempio facendo una media.

Questo approccio darà probabilmente risultati migliori di un classificatore per un singolo fotogramma, ma ignora totalmente le *dipendenze* che attraversano il video nella *dimensione temporale*.

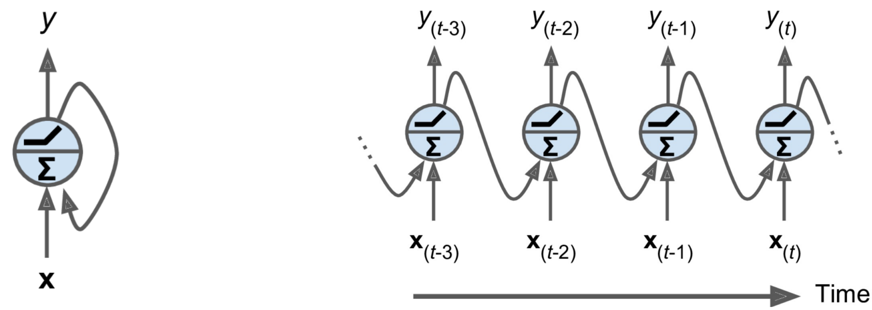
Più in generale, una vasta classe di problemi richiede di tenere in considerazione gli effetti del *tempo* su una sequenza di dati, ad esempio applicazioni

che fanno previsioni finanziarie, che generano frasi o musica o che riconoscono anomalie in sequenze di pagamenti.

Per risolvere questi problemi, la rete deve trovare dei *pattern che durino nel tempo* nelle sequenze di training. Per tale scopo vengono usate le *reti ricorrenti* (RNN), una classe di reti neurali in grado di *propagare la loro conoscenza nel tempo*, o meglio, in grado di, ricevendo in input un determinato elemento della sequenza, tenere conto anche dei precedenti.

Per queste reti, il tempo è *discreto* ed è suddiviso in *time step*, ovvero momenti che identificano l'elemento della sequenza in analisi. Se il time step t è quello attuale, $t-1$ è il time step in cui si è analizzato l'elemento precedente della sequenza e $t+1$ sarà quello in cui si analizzerà il successivo.

La RNN più semplice possibile è formata da un unico neurone che produce un output che rimanda indietro a sé stesso come ulteriore input per il time step seguente (Figura 2.18).



(a) Rappresentazione compatta. (b) Rappresentazione srotolata nel tempo.

Figura 2.18: Una semplice RNN formata da un unico neurone che propaga il suo output.

Fonte: [12]

Si può, ovviamente, avere un intero layer di questi neuroni e, ad ogni time step, far propagare il loro risultato all'indietro. Questo meccanismo richiede che per ogni livello ci siano *due set di pesi* ben distinti: quelli che vengono applicati all'input vero e proprio e quelli che sono applicati all'output del time-step precedente.

In generale, ogni parte di una rete ricorrente che preserva memoria degli stati precedenti è detta *cella di memoria*. Ogni cella produce un *output* y e uno *stato nascosto* (*hidden state*) h, entrambi in funzione del time-step t.

Lo *stato nascosto* è quello che viene passato ai time-step successivi e che, nel corso delle iterazioni, inizierà ad accumulare una visione globale della sequenza in input, mentre l'output y è il frutto di un'elaborazione istantanea (che tiene, tuttavia, conto dello stato nascosto costruito fino a quel momento).

Nella figura 2.18, output e stato nascosto sono prodotti dalla stessa funzione, ma non necessariamente è così: nel corso degli anni sono state infatti sviluppate celle sempre più complesse (come la LSTM in figura 2.19), anche per combattere due grossi problemi delle RNN: l'*instabilità* dell'addestramento e la *scarsa memoria a lungo termine* delle celle "base".

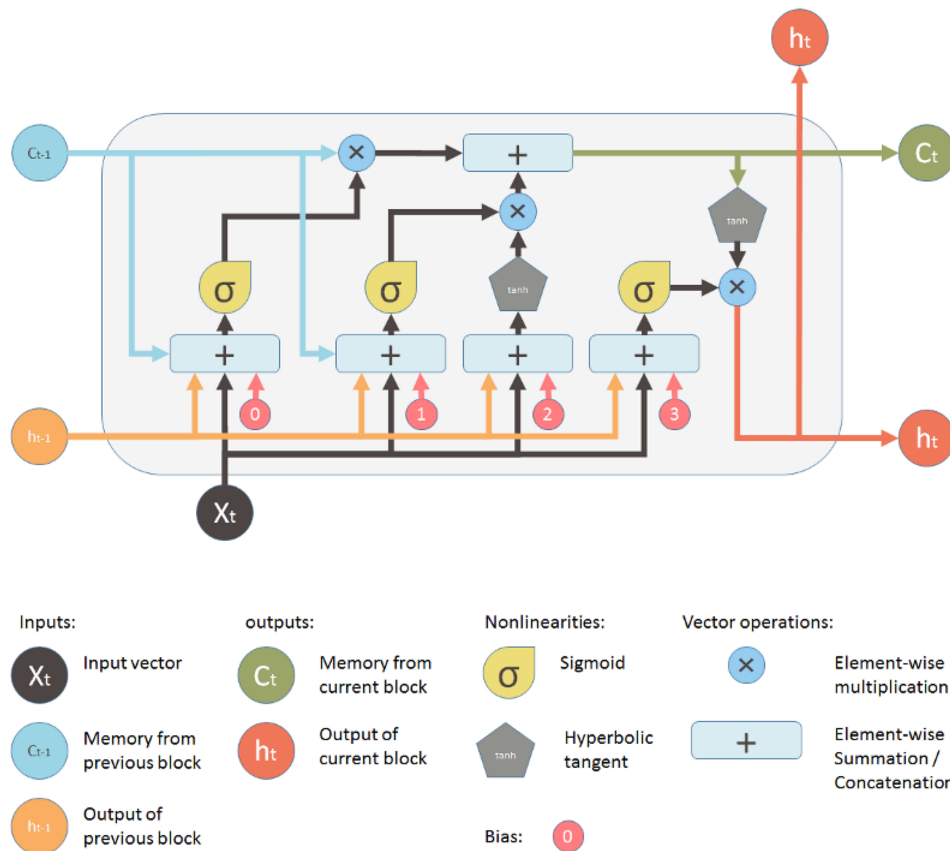


Figura 2.19: Schema di una cella LSTM, alternativa alla cella tradizionale.
Fonte: `https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714`

Il problema dell'*instabilità* nasce dalla tecnica usata per addestrare le reti ricorrenti, la *backpropagation-through-time*. In realtà, si tratta di una semplice backpropagation, ma con la rete "srotolata" nel tempo perché il gradiente deve essere fatto passare per tutti i time-step per poter cambiare i set di pesi che riguardano l'*hidden state*.

Come in una normale backpropagation, si fa entrare una sequenza di input, si considera *solo l'ultimo* output e vi si calcola la funzione di errore. A questo punto, però, il gradiente deve essere propagato nel tempo, quindi scorrere all'indietro attraverso i time-step come se attraversasse i layer di una rete convenzionale.

Questo significa che addestrare reti ricorrenti su sequenze molto lunghe equivale, in fatto di tempistiche, ad addestrare una rete classica profondissima, composta da tanti layer quanti sono i time-step, il che da origine ai problemi di vanishing/exploding gradients di cui si è parlato in precedenza.

Il problema della *brevità* di memoria è invece semplicemente legato al fatto che gli stati recenti influiscono *maggiormente* sullo stato che viene propagato nel tempo, quindi si tende a perdere traccia degli input che sono stati analizzati in un momento più lontano.

Le celle *LSTM* (Long Short-Term Memory) (Figura 2.19) sono nate proprio per cercare di risolvere questi limiti delle RNN classiche e vengono ormai usate di default in problemi che richiedono analisi di sequenze di dati, compresa l'analisi di video.

Le LSTM ricevono tre input: l'input effettivo della sequenza, l'output dal time-step precedente e la *memoria*, o meglio, una rappresentazione dello stato della sequenza nel lungo termine. La memoria (C_{t-1}) passa dapprima attraverso un *forget gate*, il quale le fa perdere alcuni "ricordi" considerati non più rilevanti. Il vettore risultante passa nell'*input gate*, che, al contrario, sceglie quali parti dell'informazione in input aggiungere alla memoria, che viene così "aggiornata". Infine, la memoria viene trasmessa al blocco di elaborazione successivo (C_t), ma allo stesso tempo fatta anche passare attraverso ad un *output gate*, che sceglie quali parti utilizzarne per mischiarla con l'input e produrre un output (h_t) nel time-step attuale.

I tre *gate* citati sono controllati sia dall'input della sequenza (X_t), che dall'output del time-step precedente (h_{t-1}) e, chiaramente, dai pesi che vi vengono applicati.

Con tutta questa elaborazione aggiuntiva, una LSTM può imparare a riconoscere e premiare le parti più importanti della sequenza in input, inserendole nella memoria a lungo termine in modo che siano tramandate alle elaborazioni successive.

2.2.5 Paradigma encoder-decoder

Una determinata classe di problemi di deep learning prevede la *trasformazione* da una sequenza in input ad un'altra sequenza in output. Questa classe di problemi è detta *sequence-to-sequence*, spesso abbreviato con la sigla *seq2seq*.

Un classico esempio di questo tipo di problemi è la *traduzione* di frasi da una lingua all'altra, dove ogni frase è vista come una sequenza ordinata di parole. Una soluzione banale per questo problema potrebbe essere un semplice *mapping* "a freddo" di ogni parola con la sua rispettiva traduzione nella lingua target. Tuttavia, in questo modo verrebbero totalmente ignorate le regole

linguistiche e il *contesto* della frase in input, rendendo la frase prodotta senza alcun dubbio sub-ottima e, con buona probabilità, errata.

Il primissimo problema da affrontare è che la lunghezza della sequenza in input è diversa da quella in output, perché le lingue hanno costrutti molto diversi tra loro. Per questo motivo è conveniente adottare una separazione in due parti del problema (Figura 2.20):

- Una rete neurale analizza l'input e ne *produce una rappresentazione* in uno spazio di dimensione fissa e ridotta (*encoder*);
- Un'altra rete *produce una sequenza* a partire da questa rappresentazione (*decoder*).

La rappresentazione intermedia viene anche detta *encoding* o *vettore di contesto* ed è, concettualmente, un "riassunto" dell'input, o meglio, una sua *rappresentazione in un iperspazio comune* alle due lingue.

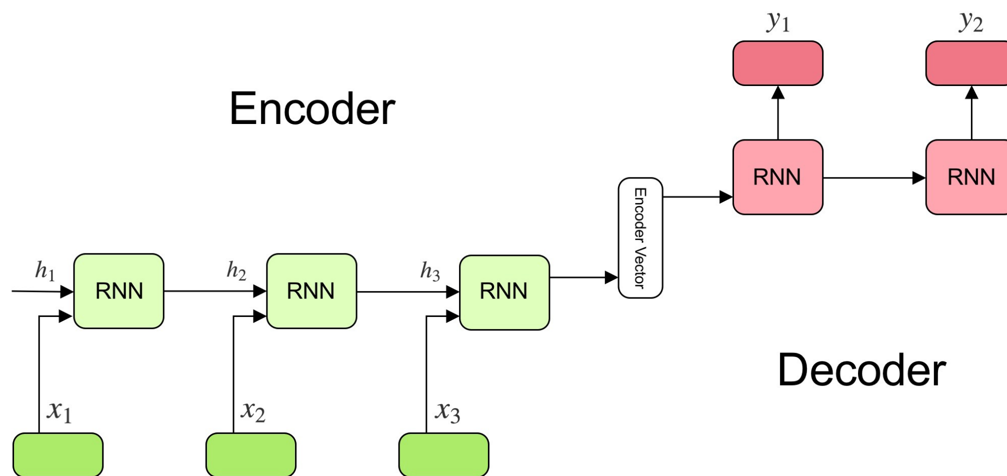


Figura 2.20: Esempio di un'architettura encoder-decoder formata da due RNN separate.
Fonte: https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

2.2.6 Meccanismi di attention

In generale si dicono *meccanismi di attention* delle pratiche in grado di *identificare* ed *avvalorare* le parti più importanti dell'informazione. Questi meccanismi donano alle reti neurali la capacità di *prestare attenzione* solo ad alcuni dettagli caratteristici piuttosto che alla visione d'insieme, la quale può essere fuorviante.

Infatti, uno dei problemi che affligge l'encoding/decoding è che ogni parte della sequenza in input viene considerata *alla pari delle altre*, quindi, in sequenze

lunghe, il vettore di contesto può risultare impreciso perché ricco di informazioni potenzialmente non rilevanti per il time step corrente del decoder.

Per fare un esempio dei benefici che porta questo meccanismo, si pensi al task di traduzione della frase "*non c'è nulla che io possa fare*" in lingua inglese, ovvero "*there is nothing I can do*". Un essere umano associa istintivamente la parola "*nothing*" alle parole "*non*" e "*nulla*" e la parola *"can"* al verbo *"potere"*: sa, quindi, identificare, per ogni parola della frase inglese le parole di quella italiana che *influiscono maggiormente* sulla traduzione.

Un decoder, tuttavia, riceve un vettore unico e *compatto* dall'encoder e non è in grado di distinguerne parti più importanti di altre per ogni time step, poiché assume che tutto quello che gli arrivi sia importante allo stesso modo per la predizione della prossima parola.

Viene quindi inserito un meccanismo di attention come *componente intermedio* tra l'encoder e il decoder in grado di imparare in autonomia ad assegnare pesi ad ognuno degli output prodotti nei time step della prima parte dell'architettura. Questi pesi sono alti per le parti rilevanti della frase e, al contrario, diminuiscono l'importanza delle parole ininfluenti.

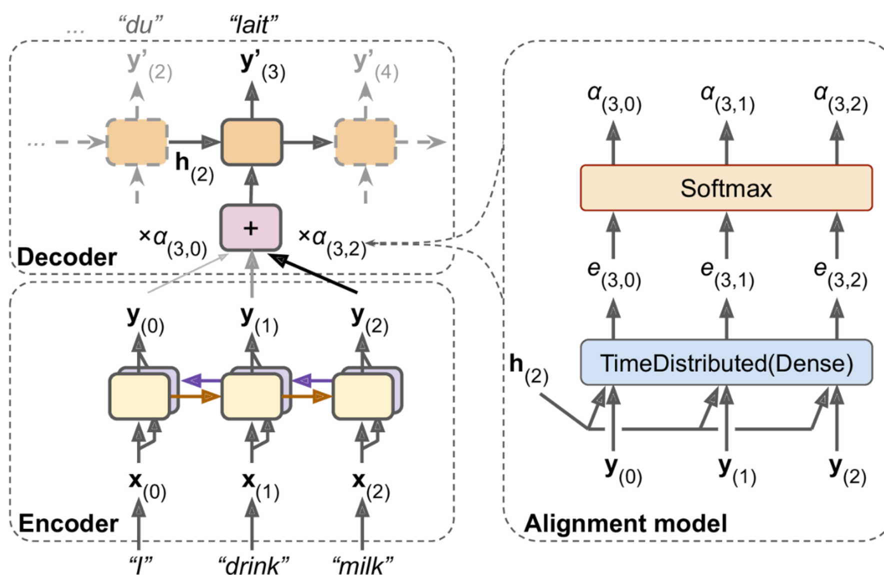


Figura 2.21: Meccanismo di attention su un modello di traduzione dall'inglese al francese.

Fonte: [12]

L'esempio in figura 2.21 mostra nella metà sinistra l'architettura encoder/-decoder di traduzione classica, ma con l'introduzione di un *modulo di attention* prima dell'inizio del decoder, espanso sulla destra.

Riprendendo l'esempio di prima, ci saranno inizialmente vettori di contesto che metteranno in evidenza il concetto di *"essere"* (*"there is"*), poi la negazione (*"nothing"*), il soggetto della frase (*"I"*), e infine i due verbi (*"can"*, in cui sarà ben presente il concetto di *"potere"* e *"do"*, in cui sarà la parola *"fare"* quella più evidenziata).

2.3 Immagini e video come dati di input

Il paradigma encoder-decoder e l'attention sono stati applicati in modo intensivo per i problemi di *machine translation*. Pochi anni fa, si è pensato di combinare le CNN e queste trasformazioni sequence-to-sequence per affrontare l'*image captioning*, ovvero il problema di creare frasi in linguaggio naturale che descrivano un'immagine.

L'idea è che, invece di una RNN, l'encoder sia una CNN, in quanto in grado di *codificare l'immagine in uno spazio a dimensionalità fissa* che possa essere passato come input al decoder. In un certo senso, si può pensare all'image captioning come al problema della *traduzione* da un'immagine al linguaggio naturale. Essendo l'input formato da un'immagine, il vettore di contesto è più propriamente chiamato *vettore di feature*.

Inoltre, anche sulle immagini è possibile applicare meccanismi di attention: si possono infatti pesare le diverse regioni di un fotogramma e quindi le feature prodotte per quelle zone. In questo caso, si parla di *attenzione spaziale* (*spatial attention*).

Uno degli studi più importanti sull'applicazione di queste metodologie è stato condotto in [1] ed è riportato in figura 2.22. Come encoder viene utilizzata una rete convoluzionale preaddestrata su ImageNet e come decoder uno strato di LSTM che pesa le feature map prodotte nelle diverse regioni dell'immagine.

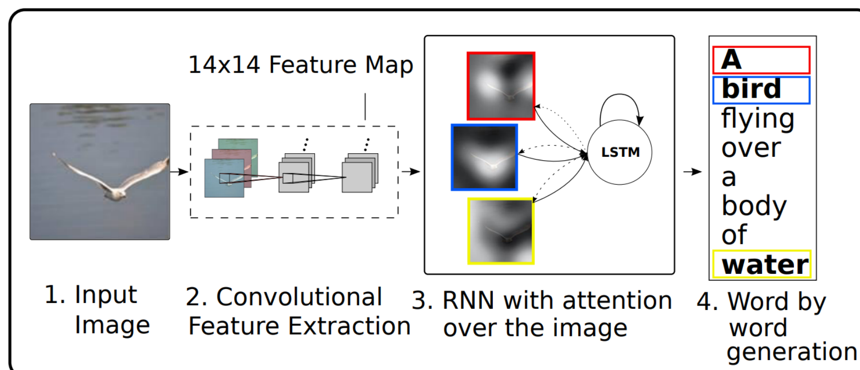


Figura 2.22: Architettura utilizzata in [1] che mostra l'applicazione del paradigma encoder-decoder e della spatial attention.

Per i video sono inoltre stati sviluppati meccanismi di *temporal attention* che, similmente a quelli per i sistemi di traduzione, avvalorano le parti rilevanti della sequenza di fotogrammi, quindi *aree temporali* del video. Questi sistemi verranno ripresi nelle prossime sezioni.

2.3.1 Architettura two-stream

Gli ottimi risultati ottenuti dall'applicazione di questo tipo di architetture sulle immagini hanno naturalmente portato i ricercatori ad adottare approcci simili anche per trattare i video. Da un lato è corretto dire che i video sono sequenze ordinate di fotogrammi, quindi i primi approcci al tipo di dato video (detti *single stream*) consistevano, molto semplicemente, nel prendere singoli frame, farli elaborare ad una CNN e fondere i vettori risultanti in vari modi [19].

Trattando i video come singoli frame sparsi non è però possibile riconoscere con efficacia un'*attività* duratura nel tempo, poiché, come si è già spiegato nella sezione 2.2.4, si ignorerebbe totalmente la dimensione temporale del video. Sarebbe, infatti, un errore assumere che i frame di un video possano essere tra loro indipendenti, in quanto ci sono forti *dipendenze temporali* che vanno colte per comprenderne a fondo il contenuto.

In generale, un video ha più canali informativi che possono essere analizzati: infatti, l'estrazione di feature rappresentative viene detta *multimodale* in quanto coinvolge i molteplici *canali* (o *modalità*) del video.

Riprendendo dal discorso iniziato nella sezione 2.2.4, una delle soluzioni per cogliere le informazioni temporali di un video potrebbe essere quella di collegare uno strato di LSTM subito dopo l'estrazione di feature del layer convoluzionale. Questo metodo permette alla rete di cogliere e trasportare nel tempo un contesto generale (dipendenze ad *alto livello*). Tuttavia, per riconoscere azioni in un video è spesso fondamentale analizzare le dipendenze con una granularità fine, a livello delle differenze che intercorrono tra i frame (dipendenze a *basso livello*).

In luce di questo, Simonyan e Zisserman in [20] propongono un'architettura detta *two-stream* per l'analisi dei video (In figura 2.23). Due reti *convoluzionali* separate vengono addestrate in contemporanea: una per il riconoscimento di immagini classico prendendo in input i singoli fotogrammi RGB (canale detto *spatial stream*) e l'altra per svolgere la parte di riconoscimento relativa allo svolgimento delle azioni, prendendo in input una sequenza di *immagini che rappresentano i movimenti* (canale detto *temporal stream*).

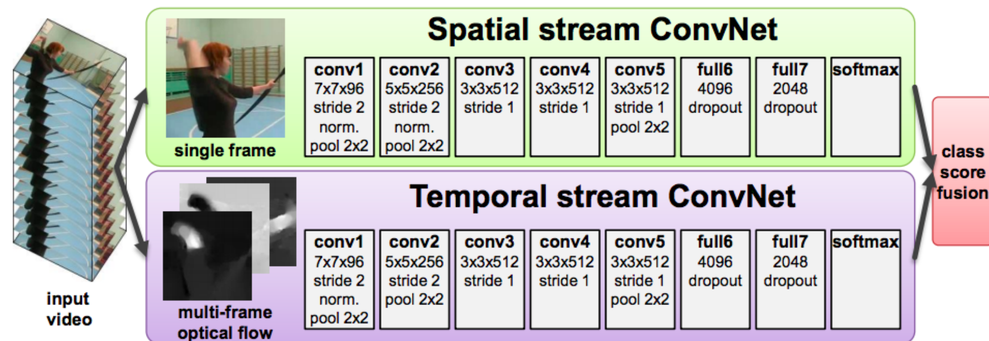


Figura 2.23: Architettura two-stream per l'action recognition in un video.
Fonte: [20]

Il funzionamento di questa architettura trova giustificazione ancora una volta nella corteccia visiva umana, in cui è presente una diramazione in due vie, o *stream*:

- La *via ventrale* (*ventral stream*) ha il compito di riconoscere forme ed oggetti presenti nell'immagine che riceve ed è quindi analoga allo *spatial stream*;
- La *via dorsale* (*dorsal stream*) è invece legata principalmente al riconoscimento dei movimenti ed è quindi tradotta nel *temporal stream*.

Per rappresentare i movimenti in questa architettura vengono usati gli *optical flow*, ovvero immagini che rappresentano lo spostamento di oggetti e superfici calcolate attraverso l'algoritmo $TV\text{-}L^1$ [21] partendo da due frame adiacenti. TV-L^1 è un algoritmo *pesante*, ma anche molto *resistente* a variazioni improvvise di illuminazione o spostamenti bruschi, al contrario della più veloce, ma anche più "sporca", differenza diretta tra i frame RGB.

2.3.2 Architettura a convoluzioni 3D

Per analizzare un video si può anche prendere una strada totalmente diversa. I filtri imparati dalle reti convoluzionali sono 2D, perché pensati per trovare pattern all'interno delle due dimensioni di cui si compone un'immagine. Forzare le dipendenze temporali tra i frame ad essere rappresentate come *immagini bidimensionali* può sicuramente dare buoni risultati, ma non è la via più intuitiva.

Si può, al contrario, immaginare un video come un'*immagine tridimensionale*, formata impilando uno sopra l'altro tutti i fotogrammi che lo compongono: la terza dimensione così creata sarà quella del *tempo*. Tutti i pattern temporali che nell'architettura two-stream vengono imparati attraverso gli optical flow

sono ora presenti in questa dimensione e possono essere scoperti utilizzando una rete convoluzionale con *filtri tridimensionali* (Fig. 2.24).

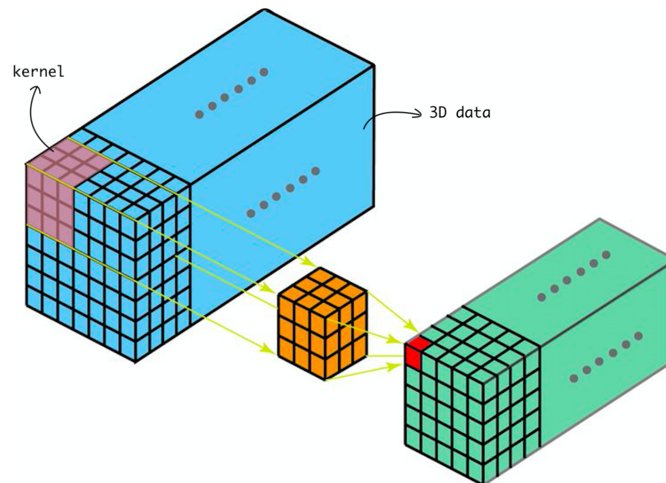


Figura 2.24: Una convoluzione 3D. A sinistra, la rappresentazione tridimensionale del video (ogni quadratino è un pixel e ogni "strato" un fotogramma), al centro il kernel della convoluzione e a destra la feature map prodotta. Fonte: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

L'elaborazione del video attraverso una 3D CNN risulta più *naturale* perché le feature temporali vengono estratte in automatico insieme a quelle spaziali e la predizione finale comprende automaticamente questi due aspetti, al contrario dell'architettura two-stream che mantiene separate le due rappresentazioni ed è costretta a unirle successivamente (o ad unire due predizioni separate) in maniera potenzialmente sub-ottimale.

Una critica che viene spesso mossa a questo tipo di reti riguarda la loro complessità e il loro enorme numero di parametri, il che le rende più lente, pesanti e a rischio di overfitting rispetto alle architetture a stream. Nel corso degli anni questi problemi sono stati parzialmente risolti tramite architetture sempre più accurate e allo stesso tempo leggere, come *I3D* [22], che verrà analizzata nel dettaglio nel successivo capitolo ed è stata, prima di essere superata da un'ulteriore architettura a convoluzioni tridimensionali, lo state-of-the-art sul dataset UCF101.

2.3.3 Altre architetture

La distinzione tra i due tipi di architetture presentate in precedenza non è, in realtà, così netta. Molto spesso i risultati migliori si ottengono costruendo modelli eterogenei e modulari usando liberamente i componenti visti finora.

Alle reti che usano convoluzioni 3D, ad esempio, viene spesso affiancato almeno un altro stream che analizza gli optical flow come nell'architettura standard two-stream per ottenere risultati più accurati.

Al centro di altri studi recenti c'è invece il *problema del pre-calcolo degli optical flow*. L'aumento di precisione che fornisce l'analisi degli optical flow rispetto a sistemi che utilizzano solamente la RGB modality è testimoniato da svariati studi, ma pre-calcolarli per ogni coppia di frame adiacenti del video è un grosso *bottleneck* per l'elaborazione a livello di complessità computazionale.

In [23] il problema viene risolto con un'architettura two-stream e una rete aggiuntiva che precede la CNN dello stream temporale. Questa rete è detta *MotionNet* e viene addestrata in maniera non supervisionata a produrre una stima dell'optical flow per il video in esame. In [24] viene fatta la stessa cosa con una piccola rete chiamata *SPyNet* (anche se, oltre a questo, per ottenere risultati ancora migliori, si affianca un terzo stream che analizza gli optical flow pre-computati tradizionalmente con TV-L^1).

Le architetture per l'analisi video si sono quindi evolute in forme molto varie e la distinzione tra two-stream e 3D CNN diventa sempre più labile.

A tutte queste scelte architetturali vengono anche ad aggiungersi le decisioni da prendere riguardanti i componenti interni: quale CNN utilizzare tra la moltitudine di quelle state-of-the-art preaddestrate disponibili? Come strutturare componenti custom come ad esempio la MotionNet citata prima? Quali meccanismi possono migliorare ulteriormente i risultati? Questa serie di domande è il motivo per cui la ricerca in questo campo è così fervente.

Capitolo 3

Analisi delle soluzioni

Si procede descrivendo più nel dettaglio i problemi da affrontare e le migliori soluzioni attualmente disponibili.

3.1 Video Captioning

Il video captioning è definito come il problema della creazione di frasi per descrivere il contenuto di un video, o più formalmente:

> *Dato in input un video V = {f1, . . . , fN} con N la lunghezza della sequenza di frame, il target del video captioning è quello di generare una frase (sequenza di parole) Y = {y1, . . . , yT} che descriva il contenuto del video.* [9]

Come abbiamo visto nella sezione 2.2.5, si può trattare il problema come una trasformazione *sequence-to-sequence* (da sequenza di frame a sequenza di parole) con il paradigma *encoder-decoder*, dove l'*encoder* è la parte che studia la rappresentazione del video e il *decoder* quella che genera le frasi.
Si possono distinguere tre fasi all'interno di questo processo:

- La fase di *estrazione delle feature* ha lo scopo di analizzare le varie modality del video (singoli fotogrammi come immagini, la traccia audio, il movimento degli oggetti tra i frame o altre feature semantiche) e rappresentare ognuno di questi aspetti come *vettori di feature*;
- La fase di *aggregazione* vede la valutazione delle diverse strategie con cui è possibile combinare i vettori di feature estratti al passo precedente in modo da ottenere una rappresentazione del video *unificata*;
- La fase di *decoding* sfrutta le reti neurali ricorrenti per generare parola per parola una frase che descriva il video.

3.1.1 Image Captioning

Un task molto simile - ma anche più semplice - è quello dell'*Image Captioning*. Si tratta della creazione di frasi per descrivere un'immagine. In figura 3.1 viene illustrata una possibile soluzione per il problema.

La fase di aggregazione è opzionale, dato che trattandosi di immagini singole è presente un'unica modalità. Tuttavia, nulla vieta che tra la fase di estrazione delle feature e quella di decoding ci siano meccanismi aggiuntivi come ad esempio quelli di *attention* spaziale già visti.

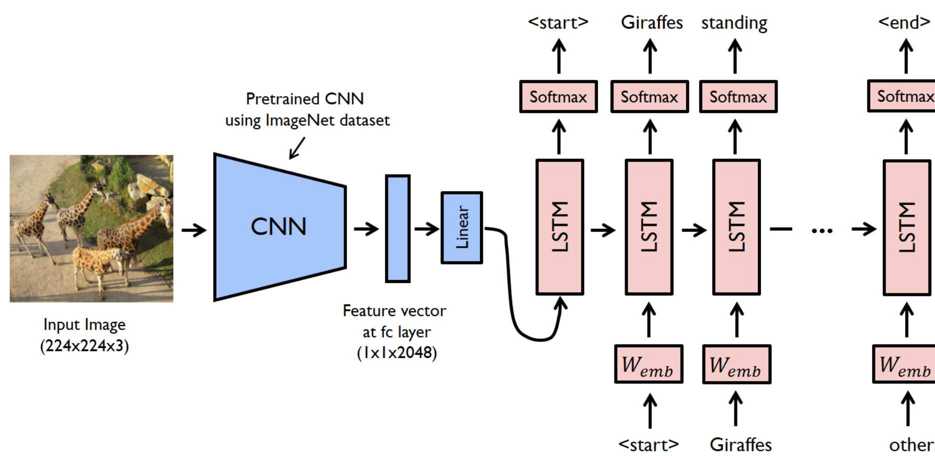


Figura 3.1: Rappresentazione di una classica soluzione per l'image captioning.
Fonte: https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/

Il decoder è invece una RNN con layer di celle LSTM che generano una frase selezionando al time step t la parola che ha probabilità più alta di essere scritta alla posizione t della frase dato il vettore di feature in input e la parola selezionata al time step $t-1$: si tratta dunque di un problema di *classificazione* basato sul calcolo di una probabilità condizionata. L'output è un vettore di numeri da 0 a 1 che indicano la probabilità per ogni parola del vocabolario.

Le soluzioni per l'Image Captioning sono un punto di partenza per quelle del Video Captioning, il quale è un problema affrontato in modi analoghi. In generale, le differenze tra le soluzioni che analizzano immagini e le corrispondenti architetture per i video sono tre:

- I contenuti di un video possono essere molto *dinamici* e tagli/cambi di inquadrature possono far variare il contesto anche all'interno dello stesso video;
- Un video non è solo l'insieme dei fotogrammi che lo compongono: vanno cioè considerate le dinamiche temporali che li collegano;

- Ci sono diverse *modality* di cui si può tener conto: le singole immagini, i canali audio, i movimenti degli oggetti e altre feature semantiche. La fase di encoding viene infatti spesso denominata *encoding multimodale*.

3.1.2 Analisi del problema

Come nell'Image Captioning, anche nel Video Captioning vengono molto spesso riutilizzate reti preaddestrate per quanto riguarda l'estrazione delle feature. A seconda delle modalità, alcuni tipi di reti sembrano in generale funzionare meglio:

- Per le informazioni riguardanti i singoli fotogrammi si utilizzano CNN come quelle già viste nella sezione 2.2.3 (*VGGNet* [17], *ResNet* [18] o *Inception* [15]). Questi sistemi state-of-the-art preaddestrati su dataset di grandi dimensioni come ImageNet garantiscono un bassissimo tasso di errore;

- Per le informazioni dal canale audio, venivano estratti ed analizzati manualmente i *coefficienti spettrali Mel* (MFCC). Di recente è stato dimostrato che anche questa analisi può essere effettuata da reti neurali come *VGGish* con una migliore efficacia e, in questo caso, si parla di *deep audio features*;

- Per l'estrazione di feature motorie vengono solitamente usate 3D CNN. Architetture state-of-the-art come *MGSA* integrano componenti custom in grado di aumentare il numero e la qualità delle informazioni catturabili dalle convoluzioni;

- Infine, le informazioni semantiche, che riguardano principalmente il contesto del video, sono spesso tralasciate e occasionalmente estratte da componenti custom estremamente specifici. *M&M TGM*, ad esempio, addestra una piccola rete neurale a riconoscere i topic latenti di un video e, una volta estratti, li include nell'input di un decoder custom che viene detto *topic-aware*.

Queste feature sono raccolte in maniera sequenziale prendendo come input tutti o una parte dei fotogrammi. In questo modo, si produce una sequenza di encoding, ognuno dei quali si riferisce ad un punto preciso del video. Questa sequenza ha una lunghezza dipendente da *quella del video* e, sebbene non sarebbe un errore utilizzare una rete ricorrente per trattarla e prenderne l'output finale come rappresentazione in uno spazio a dimensionalità ridotta del video, come in [25], bisogna considerare che *non tutte le modalità sono*

sempre equamente importanti, che avere un numero di feature elevato rischia di mandare in overfitting il modello e che, in fase di training, avere una sequenza potenzialmente molto lunga rischia di introdurre i problemi di *vanishing* o *exploding gradients*.

Al fine di formare una rappresentazione aggregata più compatta, segue quindi una fase di *pooling* della sequenza. Una scelta banale e poco performante è quella di effettuare un *average pooling* e quindi presentare al decoder una singolo vettore riassuntivo del video.

I meccanismi di attention sono in grado di assegnare pesi più significativi. Come è già stato fatto presente, l'attention può essere *temporale* o *spaziale*, cioè:

- Con attention *temporale* si assegnano pesi legati a *momenti* precisi del video, quindi elementi della sequenza di encoding. Di solito è utilizzata per rendere noto al decoder quali istanti possono influenzare maggiormente la scelta di una parola al time step in cui si trova;

- Con attention *spaziale* vengono evidenziate delle *regioni spaziali* all'interno dei frame o delle immagini di optical flow che possano risultare più importanti, come se fosse una messa a fuoco sui punti in cui avvengono azioni determinanti. Non è usata strettamente per compattare la sequenza di encoding, ma per renderla più *significativa*.

Altri meccanismi di aggregazione sono stati sviluppati per applicare attention in modo dinamico ai vari stream provenienti dall'estrazione di feature dalle diverse modalità. Questi meccanismi di aggregazione pongono il loro focus sul fondere i vettori di feature dando la giusta importanza ad ognuno.

Una volta che un vettore aggregato è stato sviluppato, questo viene passato come input al decoder, il quale ha il compito di generare la frase come visto in precedenza.

Un esempio completo di un'architettura di video captioning si può trovare in figura 3.2. L'estrazione di feature in questo caso comprende solamente le modality dei frame RGB e del movimento.

L'obiettivo comune di addestramento è quello di massimizzare la *similarità* tra la frase prodotta e quelle preesistenti. Tuttavia, questa funzione obiettivo è molto diversa sia dai parametri di giudizio di una "buona frase" umani che dalle metriche adottate in via di valutazione del modello (che sono di solito tre: *BLEU@4*, *METEOR* e *CIDEr*). Ci si riferisce a questo problema come *objective mismatch* e si cerca di farvi fronte in due modi:

- Aggiungere dei vincoli che leghino più strettamente il contenuto del video e quello delle frasi a livello di consistenza semantica. Questa opzione

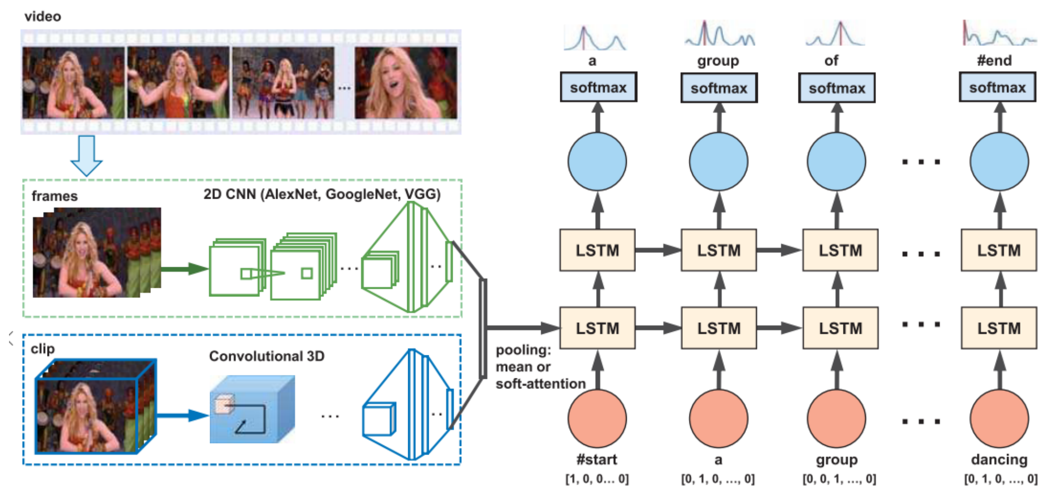


Figura 3.2: Esempio di una soluzione completa per il video captioning
Fonte: [10]

viene anche detta *supervisione semantica ausiliare* perché ha come scopo quello di produrre frasi più precise. Uno dei framework state-of-the-art attuale, *M&M TGM* [26], cerca ad esempio di predire il topic del video in modo da "specializzare" il vocabolario della frase, la quale viene costruita con un decoder detto *topic-aware*;

- Cambiare metrica di valutazione, in particolare definendo nuove metriche differenziabili e ottimizzabili via backpropagation, come le recenti *CIDEr*, *CIDEnt* o la più recente *SPICE*.

3.1.3 Presentazione dei modelli state-of-the-art

Essendo il problema del video captioning così recente, i ricercatori propongono continuamente nuove soluzioni che, pur mantenendo la struttura di base encoder-CNN/decoder-RNN che si è già rivelata vincente nell'image captioning, introducono meccanismi che ottimizzano il processo.

In [9] vengono analizzate alcune delle architetture attualmente in circolazione e riportati i punteggi ottenuti sulle varie metriche di valutazione.

Da questa analisi emerge che ancora non esiste un unico modello state-of-the-art che surclassi totalmente gli altri, ma ne esistono alcuni che performano meglio usando determinate metriche su alcuni dataset.

In questa sede si analizzano tre delle architetture con i punteggi generalmente migliori: HACA, MGSA e M&M TGM.

HACA

HACA (*Hierarchically Aligned Cross-Modal Attention*) [27] è un'architettura il cui maggior punto di forza è la parte legata alla *fusione multimodale* delle feature, oltre ad essere una delle prime ad utilizzare una rete neurale per l'analisi dell'audio (*VGGish*) piuttosto che la tradizionale estrazione manuale di tali feature. Per l'analisi visuale dei fotogrammi viene usata una ResNet preaddestrata.

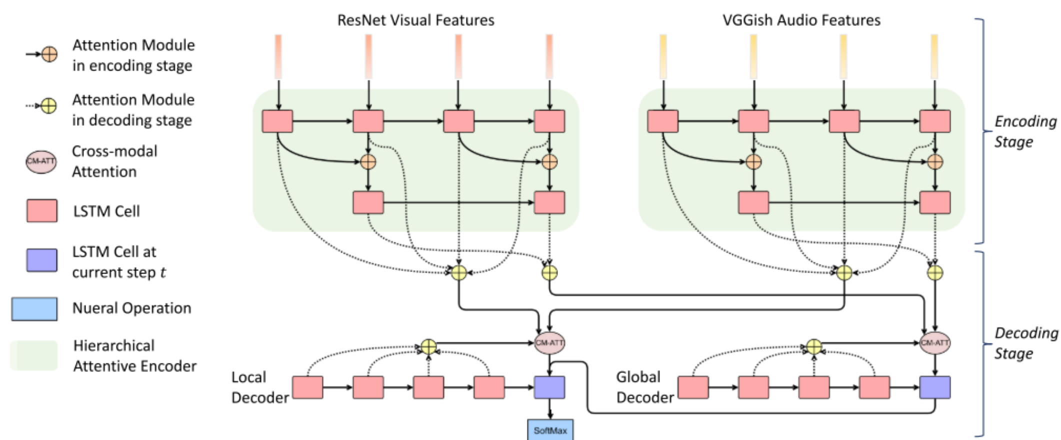


Figura 3.3: Architettura del framework HACA
Fonte: [27]

Il modello, illustrato in Figura 3.3, fa uso di moduli chiamati *Hierarchically Attemptive Encoders*, uno per modalità di interesse. Questi encoder, composti ognuno da due LSTM, sono meccanismi di attention che vengono addestrati a pesare le feature estratte dalla modalità che gestiscono e a produrre in output rappresentazioni dei contesti sia globali (high-level) che locali (low-level).

Questi contesti vengono poi utilizzati all'interno di *due diversi decoder*:

- Un decoder *globale* si occupa delle feature ad alto livello e del contesto a lungo termine del video: produce in output un vettore di *contesto globale*;
- Un decoder *locale* che riceve in input i vettori di contesto locali e il contesto globale elaborato dal primo decoder per produrre le probabilità per la prossima parola della frase.

La figura 3.3 evidenzia la struttura degli Hierachically Attemptive Encoders e la separazione che viene effettuata tra i contesti globali e locali. In questo caso vengono considerate solamente due modality (audio e immagini), ma il modello è facilmente estendibile per permettere altri stream di feature.

Diversi moduli di attention intermedi permettono di dare selettivamente un valore maggiore ad alcune modality piuttosto che ad altre: sono infatti chiamati moduli di *cross-modal attention*. Il risultato finale è quello di un sistema capace di distinguere azioni complesse e pattern nel lungo termine.

HACA detiene il punteggio più alto sul dataset MSR-VTT considerando METEOR come metrica di valutazione ed arriva a punteggi abbastanza alti anche con le altre metriche (CIDEr e BLEU@4). Questi punteggi rappresentano un enorme salto di qualità rispetto a metodi precedenti che hanno un focus analogo sulla scomposizione in feature multimodali.

MGSA

Il framework MGSA (Motion Guided Spatial Attention) [28] si concentra principalmente sullo step di *aggregazione*. L'idea alla base è quella di dare maggiore importanza agli spostamenti e alle rapide variazioni di regioni tra le immagini, cioè alla modality riguardante i movimenti. Per rappresentarli, MGSA sceglie la via degli optical flow, come si vede in figura 3.4.

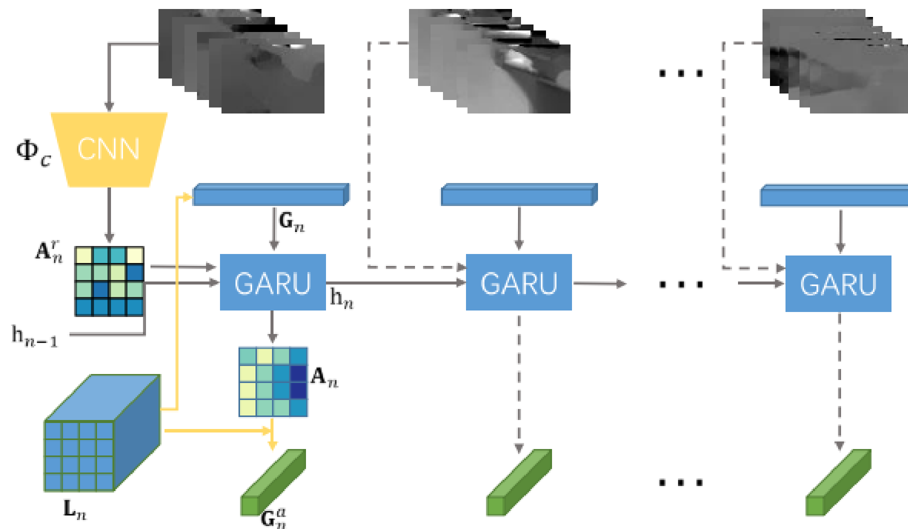


Figura 3.4: Architettura del framework MGSA

Fonte: [28]

Dal video vengono selezionati in modo uniforme una serie di fotogrammi chiave ed un insieme di frame immediatamente precedenti o successivi. Per questi, si calcolano le immagini di optical flow che vengono date come input ad una CNN preaddestrata per estrarre le feature locali come in un normale passo di encoding.

Le varie feature raccolte al primo step sono passate ad una CNN custom (Φ_c) che le aggrega applicando *attenzione spaziale* e formando una *attention map* (A_n^r). Per fare un esempio, se il video presenta un individuo che lancia un oggetto, questo passaggio darà maggiore importanza prima alla regione in cui si trova l'uomo mentre si muove nell'azione di lancio e successivamente allo spostamento dell'oggetto lanciato. La scelta di porre particolare attenzione all'optical flow è giustificata scientificamente dal fatto che anche l'attenzione umana è attratta da rapidi spostamenti.

L'attention map è già una prima rappresentazione dei punti interessanti del video, ma considera solamente l'intorno di frame del fotogramma chiave, mentre l'azione deve essere analizzata per tutto il tempo in cui viene compiuta: è cioè interessante analizzare le *relazioni* che legano le attention maps *nel tempo*.

Come ulteriore passo di processing, le attention maps prodotte sono date come input a moduli denominati *GARU (Gated Attention Recurrent Unit)*, unità ricorrenti come le LSTM, ma con lo specifico scopo di rifinire le attention maps e propagarle nel tempo.

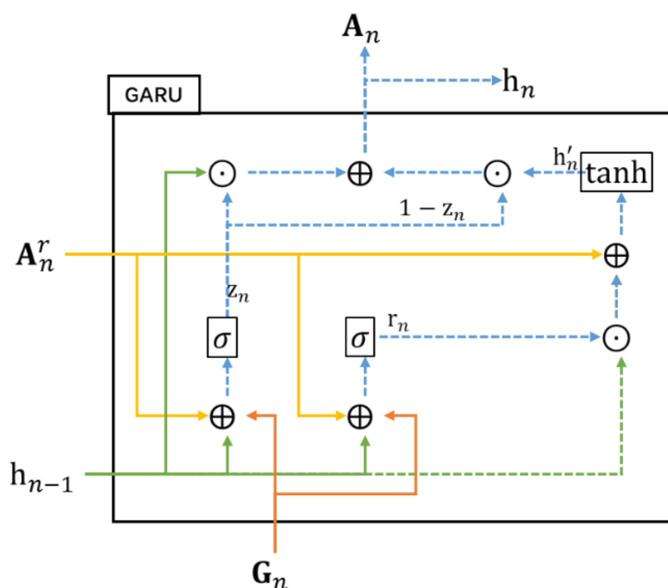


Figura 3.5: Architettura di un modulo GARU (Gated Attention Recurrent Unit)

Le GARU prendono in input anche il vettore di feature globale riguardante il fotogramma chiave (G_n): in questo modo vengono considerate anche informazioni strettamente spaziali. Il risultato finale è un'attention map rifinita (A_n) che viene usata per applicare attention spaziale sulle feature globali ed ottenere

una rappresentazione globale del frame (G_n^α). La rappresentazione globale del video G^α viene costruita accodando i vari G_n^α.

Il decoder, infine, è uno stack di due LSTM che deve tradizionalmente produrre le probabilità per ogni parola.

M&M TGM

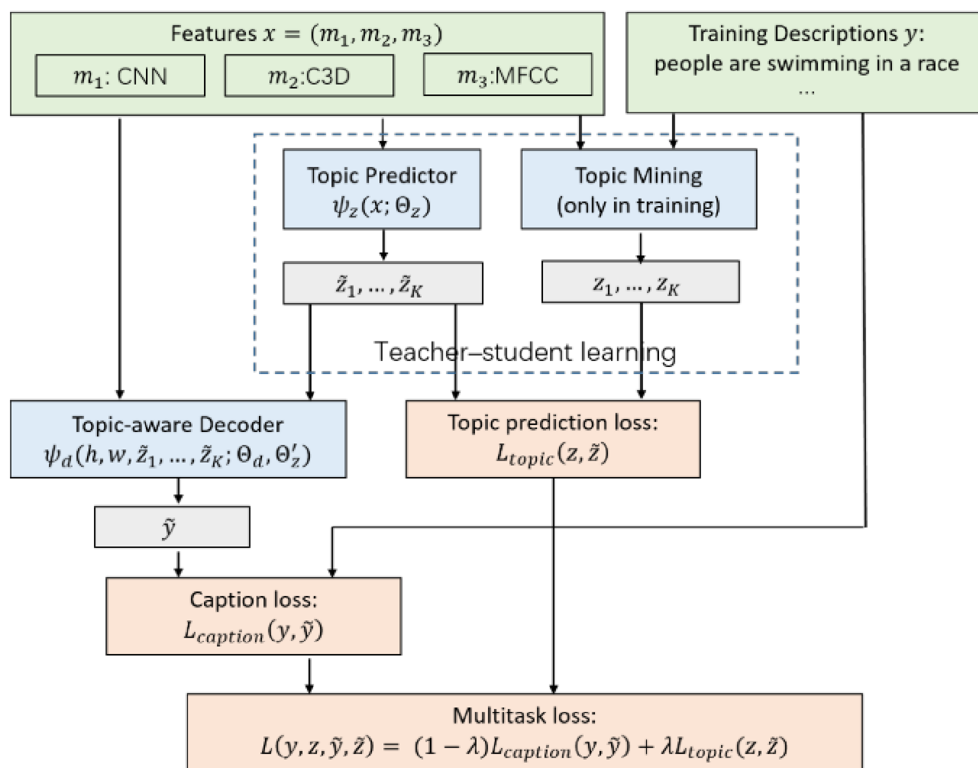


Figura 3.6: Architettura del modello M&M TGM
Fonte: [26]

M&M TGM (Multimodal latent topic guidance & Multitask learning Topic Guided Model) [26] è un framework incentrato sulla fase di *supervisione semantica*, cioè la fase opzionale in cui viene studiato il contesto del video per ottenere informazioni più precise. Sia in HACA che in MGSA si considera un vocabolario di parole molto ristretto e generale perché con una vasta scelta di parole il modello si può appesantire ed essere poco preciso. M&M TGM nasce invece dall'esigenza di "specializzare" il vocabolario usato nelle frasi a seconda del *topic* (cioè il "tema") del video ed ottenere descrizioni più dettagliate.

La fase di encoding viene sempre effettuata da CNN preaddestrate. Le feature di movimento sono estratte da una CNN 3D e le feature audio sono create a mano estraendo i coefficienti spettrali Mel.

Durante il training, vengono estratti dai video i topic in maniera non-supervisionata: questa fase viene detta *topic mining* e crea la base di conoscenza sulla quale i topic latenti dei video vengono classificati. I topic vengono estratti utilizzando sia le descrizioni testuali dei video in training che le informazioni visuali (dando maggior rilievo alle descrizioni perché meno prone a interpretazioni errate). In fase di utilizzo, le feature aggregate del video vengono passate ad un *topic predictor* che usa la conoscenza estratta in training per stabilire i topic latenti del video.

In pratica, questa fase permette di "categorizzare" il video in input e definire se si tratti, ad esempio, di un video di cucina, di sport o di una scena d'azione. Con questa conoscenza aggiuntiva, il modello può selezionare parole specifiche di tale ambito e risolvere ambiguità legate al linguaggio naturale. Per fare un esempio, la parola "frusta" ha un significato molto diverso se viene usata in ambito culinario o durante una scena di un film d'azione. Se normalmente un modello che ignora il contesto del video avrebbe dei problemi nell'utilizzo di tale parola a causa di questa ambiguità e preferirebbe parole più generali come "attrezzo" o "arma", identificare il giusto contesto permette alla rete di applicare il significato appropriato, creando di descrizioni più dettagliate.

In fase di decoding, M&M TGM usa un decoder custom per comprendere anche l'informazione aggiuntiva dei topic latenti. Questo *topic-aware decoder* prende quindi in considerazione sia il topic che le feature estratte dal video per generare una frase, risultando spesso più efficace nel fornire dettagli precisi su quello che sta succedendo.

Tutto il processo di doppia predizione è ottimizzato in modo che l'architettura, rappresentata in figura 3.6, impari in contemporanea sia i pesi riguardanti il decoder che quelli riguardanti l'estrazione dei topic: da questo la denotazione *multi-task learning.*

3.2 Nota sulle soluzioni di Video Captioning

I risultati presentati dagli articoli e dai paper riguardo alle soluzioni di video captioning sono ottimi di per sé e migliorano ad ogni nuova pubblicazione, ma nel contesto del task di riconoscimento di attività industriali ci sono diversi problemi:

- Una descrizione in linguaggio naturale *non implica la comprensione* dell'attività. Se il sistema dovesse automatizzare l'inserimento dell'attività

rilevata su un database, non riuscirebbe a dedurre una *classe* di appartenenza dalla descrizione, a meno che non si analizzi la frase prodotta con una seconda rete neurale, il che è potenzialmente dannoso dato che si possono perdere dettagli importanti presenti nel video;

- Il vocabolario utilizzato dai sistemi di video captioning è sempre piuttosto *generale*, perché la maggior parte dei dataset è composta da video con azioni semplici e quotidiane raccolte da siti di pubblico dominio. L'analisi di video industriali richiederebbe un vocabolario *specifico* e legato all'ambito di produzione, il che imporrebbe di raccogliere una grandissima quantità di dati relativi a nomi, strumenti ed azioni interessanti.

Per la maggiore attinenza al problema in considerazione, si è preferito analizzare in maniera *più approfondita* le soluzioni di action recognition. Invece di creare frasi generali riguardanti le azioni compiute nel video, l'idea diventa, quindi, quella di assegnare al video in esame uno o più label che corrispondono alle attività riconosciute.

3.3 Action Recognition

L'action recognition condivide con il video captioning i problemi riguardanti l'analisi spazio-temporale dei dati e l'estrazione dei vettori di feature, ma lo scopo finale è quello di svolgere una più semplice *classificazione* che indichi l'attività o le singole azioni che vengono svolte all'interno di un video.

Così come il video captioning trova nell'image captioning il suo antenato spirituale, anche l'action recognition ha origine dal problema di image classification, il quale è meno recente ed ha per questo ispirato un numero maggiore di soluzioni efficaci.

3.3.1 Valutazione di sistemi di classificazione

Una differenza importante tra questo problema e quello di captioning è il fatto che nell'action recognition la metrica di giudizio dell'efficacia di una soluzione è ben definita, poiché, nota la classe di appartenenza di un video, è istantaneo verificare se la previsione prodotta dalla rete risulta corretta oppure no.

La metrica più usata in generale nei problemi di classificazione è quella dell'*accuratezza*: si tratta di un valore percentuale che indica *quante* previsioni sono state *corrette* sul numero totale di previsioni effettuate sul validation/test set.

L'accuratezza non è però una misura totalmente affidabile: ad esempio, se la distribuzione dei video tra le classi del dataset non è *equa*, ovvero se ci sono poche classi con molti dati e molte classi con pochi dati, il dataset viene detto *unbalanced* (sbilanciato) e la rete viene addestrata con un *bias* verso il riconoscimento delle classi più rappresentate.

L'algoritmo diventa quindi molto capace a distinguere un piccolo numero di classi, mentre le altre vengono riconosciute con difficoltà. In fase di validazione, tuttavia, l'accuratezza potrebbe comunque essere molto alta se alla rete vengono sottoposti principalmente dati che fanno parte delle classi favorite.

Per questa ragione, per la valutazione di un sistema di classificazione si utilizza spesso la *matrice di confusione*.

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figura 3.7: Una matrice di confusione per un sistema di classificazione a due classi (*positive* e *negative*).
Fonte: https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/

Questa speciale matrice ha sulle righe e sulle colonne tutte le classi che il dataset mette a disposizione. In ogni cella viene inserito il numero di istanze della classe sulla colonna che sono state classificate come appartenenti alla classe sulla riga. Di conseguenza, sulla *diagonale* della matrice si ritrovano le istanze correttamente classificate per ogni classe, mentre gli altri valori rappresentano gli errori fatti dal modello.

In questo modo, si fornisce una vista più precisa di ciò che il modello ha imparato, in particolare evidenziando eventuali anomalie nel bilanciamento del validation set e, soprattutto, mostrando quali classi sono le più *confuse*.

La figura 3.7 mostra la denominazione delle regioni della matrice:

- True Positives (TP) indica le istanze di una classe (positive) classificate in modo corretto;

- False Positives (FP) mostra il numero di istanze dell'altra classe (negative) che sono state classificate come appartenenti alla classe in considerazione (positive);
- False Negatives (FN) indica le istanze della prima classe (positive) che sono state identificate in maniera errata (negative);
- True Negatives (TN) indica, infine, le istanze dell'altra classe (negative) che sono state correttamente classificate.

Da questa matrice è possibile ricavare diverse misure:

- L'*accuratezza* già vista è calcolabile come il numero di istanze correttamente identificate sul numero di istanze analizzate in totale, cioè:

$$\frac{TP + TN}{TP + FP + FN + TN}$$

- La *precision*, che misura quanto le predizioni finali sono accurate:

$$\frac{TP}{TP + FP}$$

- La *recall*, che indica in che misura le classi effettive sono correttamente identificate, cioè:

$$\frac{TP}{TP + FN}$$

Precision e recall sono misure complementari: tentare di aumentare una delle due causa molto spesso il calo dell'altra. Ad esempio, classificando ogni istanza come *positive* si arriva ad una recall del 100% perché non ci sarà alcun false negative: si avrà però una bassissima precision, dato che ci saranno molti false positives.

Queste due misure sono spesso riunite in modo da avere un unico punteggio: l'*F1 Score*, definito come

$$\frac{2 \times Precision \times Recall}{Precision + Recall}$$

3.3.2 Analisi del problema

Le soluzioni di action recognition seguono in modo più stretto le architetture classiche già descritte nella sezione 2.3, principalmente perché il task di classificazione non è tanto complesso quanto quello del captioning. L'estrazione

del contesto può quindi essere *meno intensiva* senza nulla togliere all'efficacia finale.

La figura 3.8 propone uno schema riassuntivo di queste architetture.

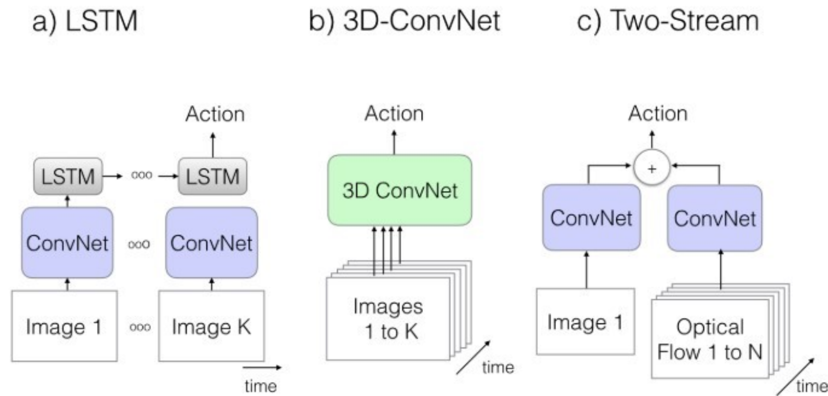


Figura 3.8: I macrogruppi di architetture per l'analisi video.
Fonte: [22]

La RNN del decoder dei problemi di video captioning è invece sostituita, nella maggior parte dei casi, da una semplice rete densa che effettua la classificazione. L'ultimo layer di questa rete ha sempre tanti neuroni quante sono le classi: ad esempio una soluzione per il dataset UCF101 avrà 101 neuroni, una per Kinetics400 400 e così via.

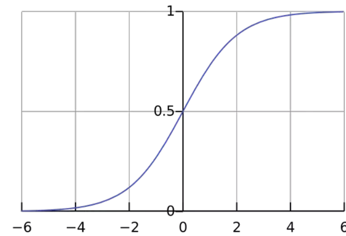


Figura 3.9: Funzione softmax.

Ogni neurone dell'ultimo layer usa la funzione di attivazione *softmax* (Figura 3.9), che, in breve, traduce i punteggi di classificazione (*logit*) in probabilità (cioè valori tra 0 e 1). Il risultato finale della classificazione sarà quindi un vettore che indica la probabilità dell'input di appartenere a ciascuna delle classi in esame, come in figura 3.10.

Qui di seguito verranno descritte nel dettaglio *quattro* architetture state-of-the-art che detengono attualmente i migliori punteggi sui dataset più importanti descritti nella sezione 1.2.1: Hidden Two-Stream e TSN per le architetture *two-stream*, I3D come esempio di architettura a *convoluzioni tridimensionali* e la recentissima SlowFast che è un'architettura 3D CNN *single stream*.

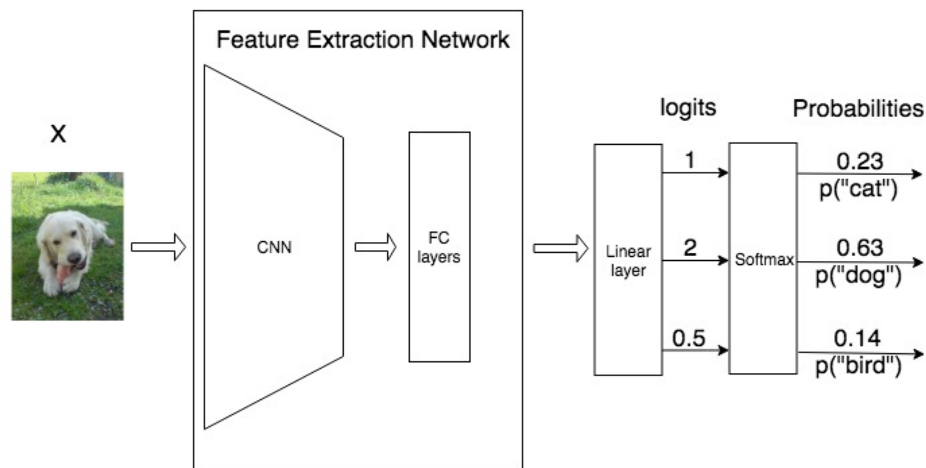


Figura 3.10: Un semplice classificatore di immagini con un layer di *softmax* per produrre le probabilità di appartenenza alle classi.
Fonte: `http://machinelearningmechanic.com/deep_learning/2019/09/04/cross-entropy-loss-derivative.html`

3.3.3 Presentazione dei modelli state-of-the-art

Hidden two-stream

Il calcolo degli optical flow tramite l'algoritmo TV-L^1 [21] è il metodo tradizionalmente usato dalle architetture two-stream per rappresentare i movimenti, ma, come già detto, si tratta di una computazione molto pesante e che richiede tantissimo spazio su disco.

Ad esempio, i video del dataset UCF101 scaricati dal sito `https://www.crcv.ucf.edu/data/UCF101.php` occupano 6,8 GB di spazio, mentre la cartella che contiene i frame RGB e gli optical flow estratti dai video tramite uno script disponibile presso `https://github.com/yjxiong/temporal-segment-networks` occupa ben 94 GB, di cui almeno i $\frac{2}{3}$ sono per le immagini di optical flow. Chiaramente, questa soluzione non scala bene e trattare dataset molto più grandi di UCF101 richiede moltissime risorse.

In aggiunta a questo, la creazione delle immagini di optical flow è fine a sè stessa, nel senso che questo tempo e spazio aggiuntivo non servono *direttamente* a migliorare l'efficacia della rete.

Il calcolo, poi, avviene tramite la computazione di un semplice algoritmo: l'optical flow è quindi una feature *manuale*, estratta alla stessa maniera per ogni video, il che la rende potenzialmente subottima.

La prima architettura presa in esame è quindi Hidden Two-Stream [23], una moderna evoluzione di *two-stream* [20] che, come già presentato nella sezione 2.3.3, fa fronte al problema del pre-calcolo degli optical flow introducendo una

piccola rete totalmente convoluzionale (MotionNet) che viene addestrata a *produrre rappresentazioni del movimento.*

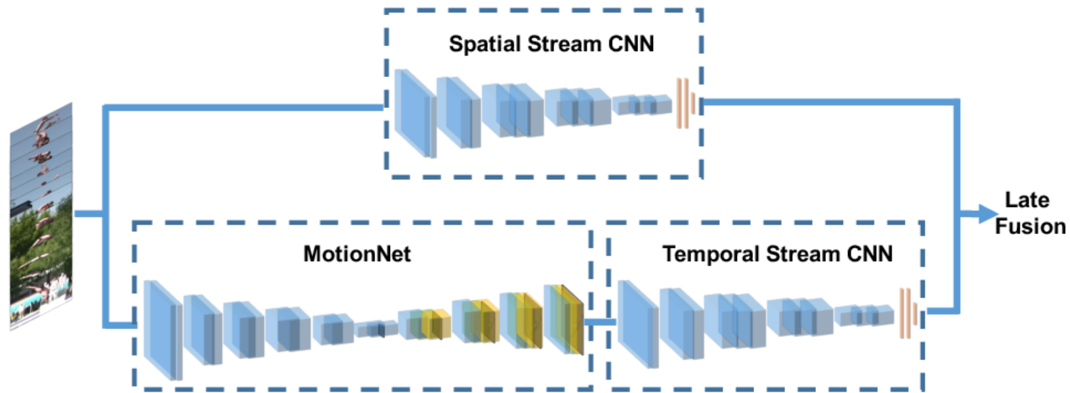


Figura 3.11: L'architettura Hidden Two-Stream
Fonte: [23]

L'architettura, mostrata in figura 3.11, mantiene la classica struttura di two-stream, con le due CNN separate che analizzano le componenti spaziali e temporali del video, seguite da una fusione dei risultati e da una rete densa per la classificazione.

MotionNet è inserita prima della temporal stream CNN e viene addestrata in maniera *non supervisionata* a produrre immagini che rappresentano il movimento. Gli autori sottolineano che MotionNet non produce optical flow *tradizionali*, cioè non impara a riprodurre esattamente l'algoritmo TV-L[1], ma estrae direttamente dai video di training un proprio algoritmo. La rappresentazione del movimento potrebbe quindi essere migliore, o meglio, più legata al task per cui la rete è impiegata.

L'addestramento di MotionNet è trattato come un problema di *ricostruzione di immagini*: data una coppia di frame adiacenti I_1 e I_2, si genera il flow V e si cerca di ricostruire I_1 usando I_2 e V e una funzione detta di *inverse warping* T ($I'_1 = T[I_2, V]$). Si può infine calcolare l'errore sulla ricostruzione confrontando I'_1 con I_1.

I kernel di MotionNet sono mantenuti volutamente piccoli in modo da dare maggior valore ai movimenti locali piuttosto che a quelli globali che spesso sono causati da *spostamenti della telecamera* nel caso dei video amatoriali di cui sono tendenzialmente composti i dataset.

Questa rete non è attualmente state-of-the-art su alcun dataset, ma ha comunque buonissimi risultati (su UCF101 è seconda solo a I3D, analizzata successivamente in questa sezione) e, soprattutto, velocizza di tantissimo i processi di training e testing, tanto da poter essere considerato un framework *real-time* per l'action recognition.

I3D

I3D (Inflated 3D ConvNet) [22] è un'architettura a convoluzioni tridimensionali che mira a risolvere molti dei problemi associati a questo tipo di soluzioni.

In primo luogo, la critica più spesso mossa alle 3D CNN è quella che ne contesta l'*altissimo numero di parametri*, il che non è un problema di per sé ma, unito alle dimensioni ridotte dei dataset tradizionalmente usati in questo ambito, facilita l'overfitting del modello.

Anche per questo motivo, le precedenti reti a convoluzioni tridimensionali hanno sempre avuto un design *poco profondo*, ovvero con pochi layer convoluzionali, il che rende limitata l'analisi in quanto più *superficiale*.

Al contrario, come si è visto nella sezione 2.2.3, il trend nella costruzione di sistemi per l'analisi di immagini è quello di avere reti convoluzionali molto profonde, in quanto permettono un'estrazione di feature più *completa*.

Per risolvere il problema dell'overfitting, I3D si avvale del pre-training sul dataset Kinetics [6], che è stato presentato nella sezione 1.2.1 come uno dei più grandi attualmente disponibili.

Avere una grande quantità di dati per l'addestramento permette la costruzione di una rete più complessa e profonda senza il rischio dell'overfitting, ma a questo punto il problema diventa quello di definire una nuova architettura appropriata.

Nella costruzione di un'architettura di tipo two-stream non ci si pone il problema, dato che lavorando su singoli fotogrammi o immagini di optical flow si possono sfruttare direttamente le potentissime CNN 2D state-of-the-art analizzate nella sezione 2.2.3. Architetture del genere con kernel tridimensionali non esistono e sarebbero necessari anni di ricerche per svilupparne.

L'idea davvero rivoluzionaria di I3D per risolvere questo problema è anche molto semplice: invece di creare una nuova architettura custom, si possono semplicemente riutilizzare le classiche CNN state-of-the-art 2D ma *gonfiandone* i kernel aggiungendo la *dimensione temporale* (da cui il nome *inflated*). I3D utilizza quindi una comune rete Inception-V1, ma con filtri tridimensionali piuttosto che 2D, come in figura 3.12.

Un enorme vantaggio di questa soluzione deriva dal fatto che i pesi del pre-addestramento delle reti CNN 2D sono *trasferibili* nella versione tridimensionale: in questo modo I3D è pre-addestrata non solo su Kinetics, ma anche su ImageNet ed è quindi perfettamente in grado di resistere ai problemi di overfitting.

L'aggiunta di una terza dimensione è inoltre un'operazione molto leggera: il numero di parametri delle reti I3D si mantiene competitivo, pur essendo sempre maggiore rispetto ai metodi two-stream-based.

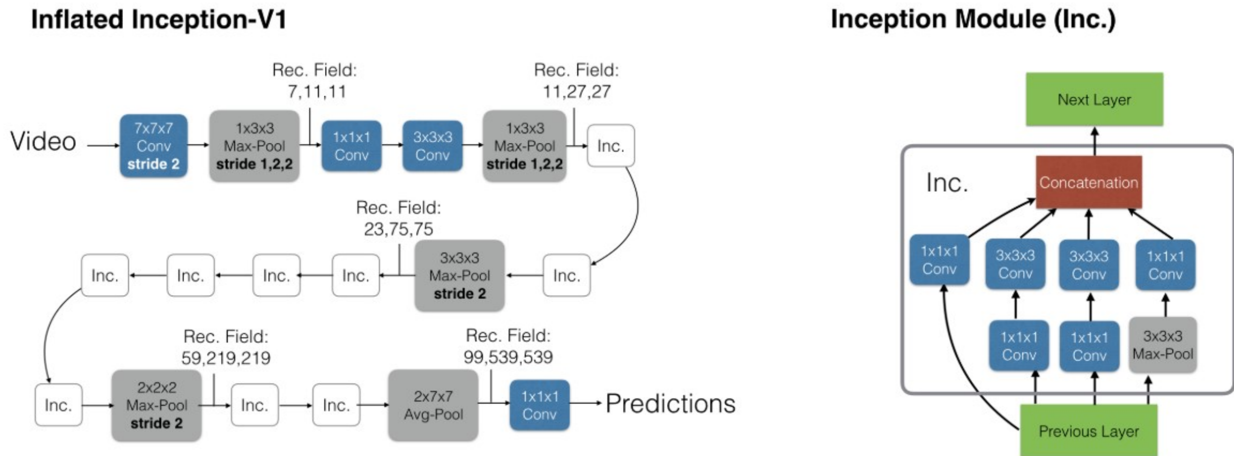


Figura 3.12: Architettura Inflated Inception-V1.
Fonte: [22]

Nonostante le CNN 3D siano pensate per lavorare esclusivamente sui frame RGB grazie alla loro capacità di estrarre in autonomia le feature temporali, gli autori di I3D notano come l'utilizzo in parallelo di un canale che analizzi tridimensionalmente anche le immagini di optical flow aumenti ulteriormente la precisione della rete.

I3D ha ottenuto i migliori risultati su UCF101 e HMDB51, mantenendo a lungo questo primato. Solo molto recentemente è stata superata da SlowFast, un'altra architettura a convoluzioni tridimensionali che verrà analizzata a breve, ma nonostante questo rimane un *milestone* di fondamentale importanza nella storia dell'action recognition e viene spesso citato come il modello che ha ridato vita alle 3D CNN.

TSN

Temporal Segment Network (TSN) [29] è un'architettura two-stream pensata per poter cogliere al meglio le dipendenze a lungo termine all'interno dei video.

Gli optical flow rappresentano le dipendenze tra un frame e quello immediatamente successivo, quindi analizzandoli uno alla volta non si può estrarre che una conoscenza estremamente *circoscritta*.

Le architetture precedenti analizzano, perciò, gli optical flow a *batch*, cioè prendendone un certo numero fisso alla volta. Le dipendenze temporali studiate in questo modo sono sempre *relativamente brevi* poiché chiaramente, per questioni di memoria, il batch deve avere una dimensione ridotta.

Se risulta più che accettabile l'estrazione di contesti brevi da video *trimmed*, in cui il soggetto non cambia così spesso e viene mostrata un'unica azione

ben definita, per i video *untrimmed* c'è assolutamente bisogno di estrarre un contesto a *lungo termine* ed è necessario farlo in modo *efficiente*.

TSN propone una strategia che permette l'estrazione di un contesto generale dall'intero video ad un costo computazionale *indipendente* dalla sua lunghezza. Questo vuol dire che con TSN si possono potenzialmente analizzare video lunghi svariati minuti senza differenze sostanziali rispetto al tempo di analisi di video brevi.

La strategia messa in atto è rappresentata in figura 3.13:

- Si scompone il video in un numero fisso di *segmenti* (di solito 7). Da ogni segmento viene estratto casualmente un piccolo *snippet* di durata fissa. Questa suddivisione è motivata dal fatto che le informazioni cambiano poco a distanza di pochi frame, mentre sezioni differenti del video possono offrire nuove prospettive sull'azione in esecuzione;

- Ogni snippet viene analizzato dalla CNN nelle sue *modality*. L'analisi produce, come da tradizione, una lista di punteggi che indicano quanto gli snippet hanno attivato ognuna delle classi target. Può essere usata una qualsiasi CNN tra quelle state-of-the-art già elencate;

- Si applica una *funzione di aggregazione* (*segmental consensus*) ai risultati dell'analisi dei vari snippet in modo da ottenere un risultato unificato sul video.

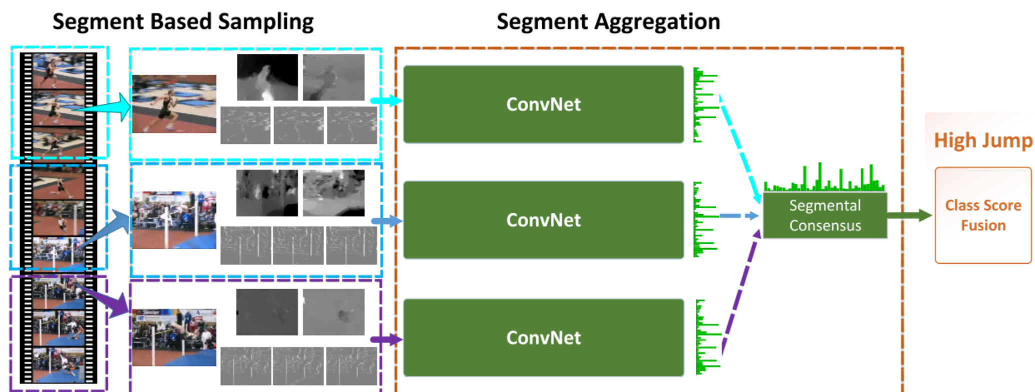


Figura 3.13: Rappresentazione del framework TSN nelle sue fasi: segmentazione del video, sampling degli snippets, estrazione di features, aggregazione dei risultati e predizione delle probabilità.

Fonte: [29]

Più formalmente, il processo è il seguente:

$$TSN(T1, T2, ..., TK) = H(G(F(T1; W), F(T2; W), ..., F(TK; W)))$$

dove $F(Ti;W)$ rappresenta una CNN con parametri W che opera sullo snippet Ti, G è la funzione di aggregazione e H è la funzione con cui si predicono le probabilità di appartenenza alle classi (ad esempio, softmax).

Gli autori esplorano l'estrazione di altre possibili informazioni in fase di pre-processing rispetto ai classici fotogrammi RGB e agli optical flow. In particolare, i migliori risultati vengono raggiunti unendo RGB, optical flow e *warped* optical flow, un algoritmo simile ma più resistente ai movimenti improvvisi della telecamera.

Viene considerata anche l'*RGB difference*, il risultato della sottrazione pixel per pixel tra due fotogrammi, il quale è un dato estremamente più veloce da ottenere rispetto al calcolo dell'optical flow. Con solo i fotogrammi e la loro differenza, TSN riesce ad affermarsi come state-of-the-art tra le architetture real-time.

Per quanto riguarda la funzione di aggregazione, nel paper vengono analizzate diverse proposte:

- Max pooling: per ogni classe viene preso il punteggio più significativo che ha ottenuto tra tutti gli snippet. Questa scelta è particolarmente *discriminante*, perché viene data enfasi ad un unico snippet per classe, perdendo le importanti informazioni che possono essere raccolte dagli altri;

- Average pooling: per ogni classe, il punteggio finale è la media di quelli ottenuti in tutti gli snippet. In questo modo viene data *uguale importanza* a tutti gli snippet, il che può essere un bene in video brevi in cui gli snippet hanno un'alta probabilità di essere abbastanza rappresentativi del video, ma può anche essere un problema qualora essi siano poco significativi (spostamenti rapidi di telecamera, parti del video scure, ...);

- Top-K pooling: l'idea è quella di mettere insieme max e average pooling, selezionando solo i K (di solito 5) snippet con i punteggi migliori per ogni classe e facendo la loro media. In questa maniera, si valorizzano le parti davvero importanti del video senza troppa complessità aggiuntiva;

- Linear weighting: invece di fare una media, si cerca di far imparare alla rete dei pesi appropriati per ogni snippet e si performa un'operazione di *media pesata*. La funzione obiettivo viene chiaramente ridefinita per comprendere la calibrazione di questi pesi;

- Attention weighting: il problema del linear weighting è quello di essere *indipendente dai dati*, poiché i pesi sono legati direttamente alla rete. Con l'attention weighting si cerca invece di assegnare un peso ad ogni snippet *variabile* da video a video a seconda del suo contenuto.

Per dataset di video trimmed, come UCF101 o Kinetics, si usa principalmente l'*average pooling* per velocizzare le operazioni, dato che non è necessario estrapolare contesti estremamente lunghi. Al contrario, sui dataset untrimmed l'*attention weighting* produce i risultati migliori.

SlowFast

Tra tutti i modelli finora presentati, SlowFast [30] è il più recente e i suoi risultati hanno straordinariamente superato lo state-of-the-art I3D.

Come le architetture a convoluzioni tridimensionali, l'idea è quella di estrarre le feature spaziali e le dipendenze temporali esclusivamente dai frame RGB.

Tuttavia, SlowFast si discosta profondamente dalle CNN 3D classiche, che implicitamente trattano spazio e tempo *allo stesso modo*, proponendo una *separazione* dell'analisi di questi due aspetti. Infatti, i *soggetti fisici*, quindi le dipendenze *spaziali*, all'interno di un video tendono a variare *lentamente*, mentre i *movimenti* e le azioni possono evolversi molto più *rapidamente*.

Per accomodare questa *dualità di scala* viene creata una rete single-stream con due *due diversi percorsi* o *pathways* (in Figura 3.14):

- La Slow pathway cattura le informazioni *semantiche* operando a un *basso frame-rate*;
- La Fast pathway studia i *movimenti* operando a frame-rate più alti.

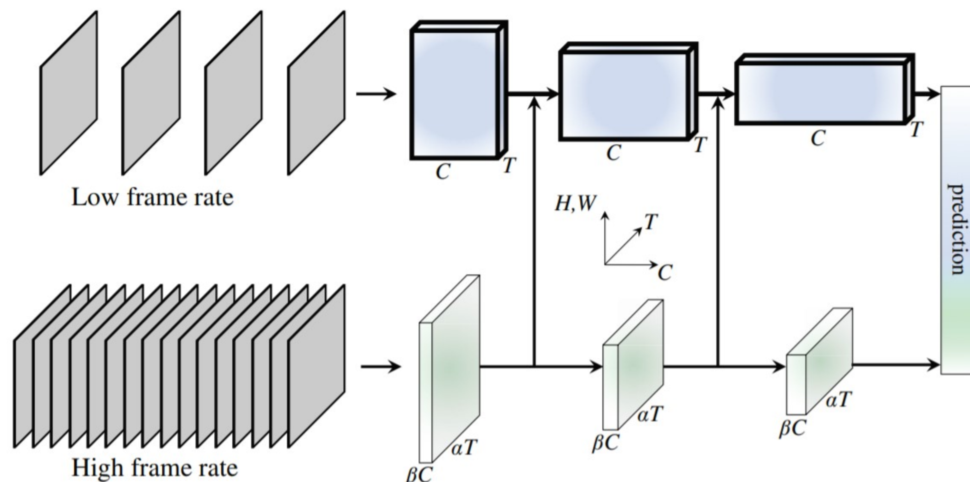


Figura 3.14: Architettura SlowFast e i suoi due percorsi.
Fonte: [30]

Il costo computazionale della Fast pathway è in realtà molto più basso del costo della Slow nonostante il maggior numero di frame che deve analizzare, poiché alla rete Fast viene ridotta la capacità di processare informazioni spaziali diminuendo il *numero di canali* in input (ad esempio, abbassando la risoluzione o rimuovendo i colori). Al contrario, la Slow pathway pone il suo focus su questi aspetti dei video, in modo da essere complementare all'altro percorso.

SlowFast può quindi sembrare simile ai metodi two-stream già visti in precedenza, ma la differenza sta nella *specializzazione* delle reti adottate. Le architetture two-stream usano spesso la stessa CNN per entrambi gli stream, quindi l'infrastruttura deve essere abbastanza *generale* per permettere una buona modellazione dei due *diversi tipi di dato*. SlowFast ha invece due infrastrutture *specializzate* ad estrarre aspetti diversi dello *stesso tipo di dato* e risulta quindi più efficace.

Inoltre, SlowFast eredita dalle architetture a convoluzioni 3D l'idea di evitare gli optical flow, in quanto feature *manuali* e molto costose da ottenere.

Le reti convoluzionali nelle due pathways sono 3D CNN qualsiasi in grado di gestire alcuni iperparametri aggiuntivi:

- τ indica quanti frame vengono ignorati prima di processarne uno nella Slow pathway. Viene anche detto *temporal stride*. Un buon valore ottenuto sperimentalmente per questo parametro è 16;

- α è il rapporto tra i frame-rate a cui operano i due percorsi ($\alpha > 1$, 8 è il valore tipicamente assegnato negli esperimenti);

- β è il rapporto tra i canali in input nei due percorsi ($\beta < 1$, $\frac{1}{8}$ è il valore classico).

Sono anche presenti delle connessioni laterali tra i due percorsi che permettono ad alcune feature estratte nella Fast pathway di essere fuse con quelle della Slow prima di alcuni layer particolarmente importanti. Alla fine dei due percorsi, le feature vengono fuse globalmente con un *average pooling* e viene effettuata la classificazione con una rete densa.

3.4 Osservazioni finali

Le soluzioni analizzate in questo capitolo sono *attualmente* alcune delle migliori disponibili. La ricerca in questo campo è tuttavia molto prolifica e soluzioni ancora migliori vengono studiate e pubblicate ad altissima frequenza, complice anche la freschezza dell'interesse verso questo tipo di dato.

Nella scelta di quali architetture presentare si sono privilegiate soluzioni con codice disponibile pubblicamente e/o citate in più paper/articoli possibili.

Chiaramente, questo non può coprire tutte le soluzioni migliori ed è possibile che non sia stata data abbastanza attenzione ad alcune delle più recenti, ma riteniamo di aver fornito una buona panoramica delle principali metodologie generali con cui il problema è correntemente affrontato.

Capitolo 4

Risultati degli esperimenti

In questo capitolo si elencano gli esperimenti svolti utilizzando le architetture e i dataset descritti finora e se ne discutono i *risultati*.

La fase di sperimentazione si è concentrata sui modelli di *action recognition*, principalmente perché più coerente con l'idea di utilizzare questi strumenti in ambito industriale per il riconoscimento di attività.

Nella seconda parte del capitolo, si introduce e valuta il nuovo dataset artificiale *blender-industrial* che rappresenta il nostro contributo personale in questo campo.

4.1 Valutazione dei modelli

Per comprendere al meglio l'efficacia dei modelli di action recognition e le loro differenze si è voluto procedere a valutarne l'*accuratezza*, addestrando alcuni di essi sui più famosi dataset già citati e sottoponendogli i relativi *validation set*.

4.1.1 Tecnologie utilizzate

Il campo dell'analisi video è molto complesso e le architetture neurali analizzate in precedenza sono estremamente difficili da costruire partendo da zero.

Esistono diversi *framework* utilizzabili in Python che facilitano la progettazione di reti neurali, tra cui Keras [31], Pytorch [32] e Caffe [33]. Alcuni dei modelli citati in questo lavoro di tesi sono stati implementati utilizzando uno o più tra questi framework e sul sito GitHub è possibile trovarne il codice liberamente disponibile.

L'utilizzo di framework appositi non solo facilita lo sviluppo delle reti, ma fornisce supporto per le fasi di training e testing delle stesse, in particolare

gestendo in automatico il *parallelismo* delle computazioni su multiple GPU che è un aspetto importantissimo in grado di velocizzare enormemente i calcoli.

Normalmente si richiede che l'ambiente di sviluppo sia Unix-based e che le GPU supportino CUDA, ma framework come Caffe forniscono un'implementazione alternativa per Windows e in OpenCL per componenti AMD o Intel, anche se attualmente si tratta di *beta* di cui non è garantito il funzionamento.

La maggior parte dei test è stata svolta su un server dotato di due GPU NVIDIA TITAN Xp con supporto fino alla versione 10.0 di CUDA e 12 GB di VRAM, su sistema operativo Ubuntu 16.04.

Alcuni test preliminari sono stati svolti su Google Colab (`https://colab.research.google.com/`), uno strumento Google gratuito che mette a disposizione uno spazio su cloud in cui poter eseguire codice Python con il supporto di svariate librerie per il machine learning. Colab permette di utilizzare anche una GPU Tesla T4 con 16 GB di VRAM.

Questo strumento ha però due problemi gravi che hanno complicato queste prime fasi: in primis, l'ambiente viene resettato ogni *otto ore*, perciò non è possibile sostenere lunghe sessioni di training o testing. Questo problema è facilmente sorvolabile con l'utilizzo dei *checkpoint*, salvataggi automatici periodici dello stato dell'addestramento del modello che permettono facilmente di riprendere il training da un punto avanzato.

Il vero problema di Colab è però la piccola disponibilità di spazio, in cui è impossibile scaricare i dataset più grandi o salvare le feature estratte come i frame RGB e gli optical flow. Google permette di collegare il proprio *spazio cloud personale* associato all'account e, con un servizio a pagamento, è possibile espanderne la capacità fino a 2 TB, ma avere un disco su cloud rende davvero lente le operazioni di I/O e, soprattutto, il salvataggio effettivo dei dati su cloud è *asincrono*, perciò non è garantito che allo scadere delle otto ore i dati siano stati effettivamente salvati.

Per questi motivi, si è reso necessario l'utilizzo del server esterno sul quale gli esperimenti sono stati svolti in remoto tramite una connessione SSH.

Onde evitare perdite di tempo nel *building delle dipendenze* dei modelli, e per permettere uno sviluppo più agevole e controllato, si è utilizzato in maniera *estensiva* Docker.

Docker è un tool che permette, come una *virtual machine*, di eseguire delle applicazioni in uno spazio virtuale detto *container*. Al contrario di una virtual machine, che *emula* in tutto e per tutto un *sistema operativo virtuale*, un container Docker contiene solamente le librerie e le dipendenze *necessarie* per l'esecuzione dell'applicazione ed è perciò molto più leggero (Figura 4.1).

L'applicazione e le sue dipendenze sono contenute all'interno di un'*immagine*, un file liberamente distribuibile ed utilizzabile su qualsiasi computer in cui

sia installato Docker, a prescindere dalle impostazioni del sistema operativo sottostante.

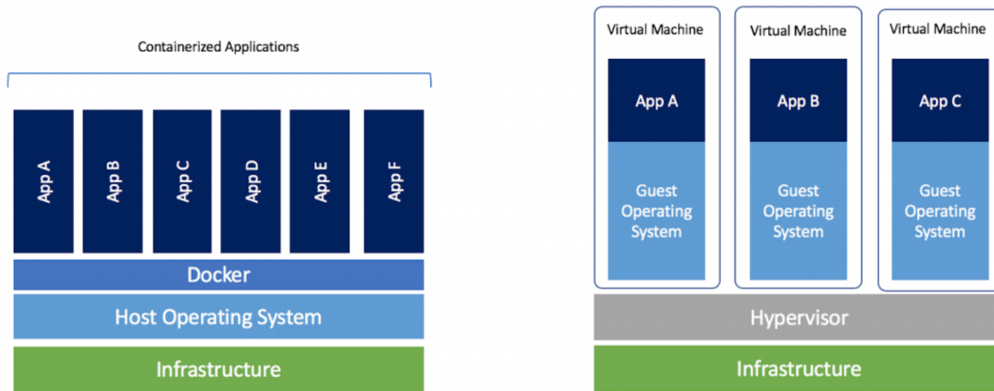


Figura 4.1: Differenze tra l'utilizzo di virtual machine (a destra) e container Docker (a sinistra) per l'esecuzione di più applicazioni.
Fonte: `https://www.docker.com/blog/containers-replacing-virtual-machines/`

I test si sono quindi principalmente svolti utilizzando l'immagine Docker del framework mmaction [34], un progetto degli stessi autori di TSN basato su PyTorch che contiene una serie di script e modelli pre-addestrati che facilitano testing e training su alcuni dei principali dataset, con il supporto per dataset custom. Il framework è un progetto in via di sviluppo e vengono inseriti nuovi modelli e dataset quasi a cadenza mensile, quindi si consiglia fortemente agli interessati di seguirne gli aggiornamenti.

La documentazione di mmaction è molto ben fornita e ci sono alcuni script che automatizzano sia il download che il preprocessing dei dataset video. Tuttavia, per avere una maggiore compatibilità con altri eventuali framework, alcuni dataset sono stati scaricati usando altri strumenti:

- UCF101 è stato scaricato dal sito ufficiale (`https://www.crcv.ucf.edu/data/UCF101.php`) e pre-processato dagli script della versione Caffe di TSN, disponibile presso `https://github.com/yjxiong/temporal-segment-networks`;
- Si è usato lo stesso procedimento per il dataset HMDB51;
- Il dataset Kinetics400 è stato scaricato utilizzando un crawler *non ufficiale* disponibile presso `https://github.com/jremmons/ActivityNet`. A differenza del tool ufficiale, il quale scarica l'intero video e poi ne mantiene solo i circa dieci secondi richiesti dal dataset, lo strumento utilizzato scarica direttamente solo la parte utile del video. Questo piccolo accorgimento

ha reso il download del dataset estremamente più veloce, in quanto alcuni snippet di Kinetics sono brevi azioni tratte da video molto lunghi che non avrebbe senso scaricare nella loro versione integrale;

- Da Kinetics400 sono stati estratti solo i frame RGB per motivi di spazio. L'estrazione è avvenuta con i tool di mmaction.

4.1.2 Analisi dei risultati ottenuti

Il primo dei test svolti è stato effettuato su Google Colab: utilizzando un'implementazione in Keras di I3D si è richiesto al modello di classificare, usando il vocabolario di Kinetics400, alcuni video personali. Il notebook è disponibile presso `https://github.com/volpepe/i3d_keras/blob/master/Keras_I3D_test.ipynb`.

Questo veloce esperimento ci ha permesso di avere una prima impressione sull'accuratezza e le performance degli strumenti attualmente disponibili: in particolare, abbiamo confermato i dubbi espressi dai ricercatori riguardo agli eccessivi tempi di estrazione degli optical flow per i video.

Con il supporto del server del Prof. Moro, si è passati alla fase di testing vera e propria utilizzando i dataset e modelli presentati negli scorsi capitoli e, principalmente, il framework mmaction. Per SlowFast, in particolare, si è usata la variante SlowOnly, formata solamente dalla Slow pathway, dato che la doppia pathway non è stata ancora implementata dal framework.

Per addestrare e/o testare i modelli tramite il framework mmaction devono essere effettuati i seguenti passi:

- Il dataset deve essere pre-processato, ovvero vanno utilizzati degli script per estrarre i frame RGB ed eventualmente l'optical flow dai video;

- Deve essere generata una *filelist*, ovvero un semplice documento di testo che indica, per ogni video, il percorso nel filesystem per raggiungere i fotogrammi, la classe di appartenenza e il numero di frame che possiede. Nel framework sono forniti script diversi per ogni dataset che generano questo tipo di file: questo vuol dire che per addestrare e testare i modelli su un dataset proprio è necessario anche scrivere il corrispondente generatore di filelist;

- Se non è fornito dal framework, va scritto il *file di configurazione* specifico per l'uso di un modello su un determinato dataset. Il file deve contenere tutti gli *iperparametri* di learning e le impostazioni necessarie agli script per gestire l'esperimento (ad esempio, il tipo di dato, i percorsi su filesystem per raggiungere i training e validation set, il numero di epoch per il training e così via).

Accuratezze percentuali top-1 / top-5 rilevate dai test svolti (pre-training su ImageNet)					
Dataset	Split	Modality	TSN	I3D	SlowOnly
UCF101	Split 1	RGB	86.55 / 97.81	*80.07 / 93.58*	*79.09 / 93.66*
		Flow	87.71 / 97.65	- / -	- / -
		Comb.	93.75 / -	- / -	- / -
	Split 2	RGB	*85.62 / 96.95*		
		Flow	*82.81 / 96.65*		
		Comb.	94.20 / -		
	Split 3	RGB	*86.53 / 96.59*		
		Flow	*81.22 / 95.21*		
		Comb.	94.07 / -		
	Avg. 3 Splits	Comb.	94.00 / -		
HMDB51	Split 1	RGB	*53.73 / 84.64*	***44.38 / 73,53***	***49.08 / 77.06***
		Flow	*49,48 / 79,02*	- / -	- / -
	Avg. 3 Splits	Comb.	68.19 / -	- / -	- / -
Kinetics400		RGB	68.95 / 87.82	72.15 / 90.19	74.73 / 91.27

Tabella 4.1: Tabella riassuntiva dei test svolti. I valori sono le accuratezze percentuali top-1 e top-5 dei modelli sui dati di validation dei vari dataset. I test preceduti da un training sono evidenziati in *corsivo*, e in particolare quelli che hanno richiesto più epoch di addestramento sono evidenziati in ***grassetto***. I valori sulle righe "Comb." sono stati ottenuti dalla versione Caffe di TSN, che calcola l'accuratezza combinando i due stream.

Accuratezze percentuali top-1 / top-5 dichiarate nei paper (pre-training su ImageNet)				
Dataset	Modality	TSN [29]	I3D [22]	SlowOnly [30]
UCF101	RGB	86.4 / - (Avg. 3 Splits)	84.5 / - (Split 1)	- / -
	Flow	90.1 / - (Avg. 3 Splits)	90.6 / - (Split 1)	- / -
	Comb.	94.9 / - (Avg. 3 Splits)	93.4 / - (Split 1)	- / -
HMDB51	RGB	- / -	49.8 / - (Split 1)	- / -
	Flow	- / -	61.9 / - (Split 1)	- / -
	Comb.	71.0 / - (Avg. 3 Splits)	66.4 / - (Split 1)	- / -
Kinetics400	RGB	- / -	71.1 / 89.3	72.6 / 90.3 [1]

Tabella 4.2: Tabella dei risultati top-1 / top-5 dichiarati dai paper.

[1] Risultati della versione con stride temporale 16: noi abbiamo usato uno stride temporale di 8.

Nella tabella 4.1 si mostrano i risultati dei test effettuati sul server, mentre in 4.2 sono presenti le accuratezze dichiarate nei vari paper con cui si possono confrontare i nostri risultati.

In particolare, dato che i modelli testati da noi sono tutti pre-addestrati su ImageNet, i risultati dei paper riportati nella tabella 4.2 sono quelli in cui il modello è stato pre-addestrato solo su tale dataset. Di conseguenza, non sono necessariamente i migliori: ad esempio, la miglior versione di I3D supera TSN sia su UCF101 (raggiungendo il 97.8% di accuratezza) che su HMDB51 (con l'80.9%), ma tale versione è pre-addestrata anche su Kinetics, il che renderebbe incoerente un confronto con i nostri valori.

Non tutti i test sono stati svolti utilizzando i modelli pre-addestrati forniti dal framework (disponibili presso `https://github.com/open-mmlab/mmaction/blob/master/MODEL_ZOO.md`). Quelli che hanno richiesto un training prima della valutazione sono indicati nella tabella 4.1. In questi casi, indipendentemente dalla complessità del modello e del dataset, si è scelto di svolgere 80 *epoch* di addestramento, principalmente per capire la velocità di stabilizzazione dei parametri.

I file di configurazione per UCF101, tranne quello per TSN-Split 1, e per il dataset HMDB51, sono stati creati manualmente basandosi su quelli già presenti ed è per questo ragionevole pensare che ci sia un certo margine di miglioramento per la fase di training.

Confrontando queste tabelle si può osservare che, nei test in cui siamo stati costretti ad addestrare il modello, le 80 epoch non sono bastate a farlo convergere, in quanto è ben visibile una certa differenza tra i risultati. Questo vale, ad esempio, sia per I3D su UCF101 e HMDB51 che per il temporal stream di TSN, dato che l'accuratezza rilevata sugli optical flow è molto più bassa di quella sui fotogrammi RGB, al contrario di quello che viene provato nel paper originale [29] e nei valori della stessa split 1 per cui il modello era pre-addestrato.

I risultati su HMDB51 sono quelli più deludenti e rendono evidenti come la mancanza di una giusta quantità di dati di training sia disastrosa se unita ad un basso numero di epoch. Nel caso di I3D e SlowOnly sono state svolte altre 40 epoch di addestramento (120 epoch totali) per verificare quanto potessero migliorare questi modelli sul dataset. L'accuratezza top-1 / top-5 è passata da 43.59% / 72.81% a *44.38% / 73.53%* nel caso di I3D e da 42.61% / 71.83% a *49.08% / 77.06%* nel caso di SlowOnly che ha evidentemente beneficiato maggiormente di questa elaborazione aggiuntiva.

I risultati su Kinetics400, invece, non si discostano troppo da quelli attesi. I modelli sono stati testati solo sui fotogrammi RGB perché l'estrazione degli

optical flow avrebbe richiesto una quantità inaccettabile di tempo e spazio. Sono stati inoltre rimossi dal validation set *157 video* che venivano mal interpretati dallo script di generazione della filelist causando errori, ma questo non dovrebbe cambiare di molto l'accuratezza finale poiché rimangono validi ben 17,857 video.

4.2 Costruzione di un dataset artificiale

Gli argomenti trattati finora sono stati esposti nel modo più generale possibile, in modo da offrire una panoramica del problema e dello stato dell'arte delle soluzioni *indipendente* dall'applicazione effettiva.

Tuttavia, come si è già spiegato parlando degli obiettivi di questa tesi nella sezione 1.3, si è anche pensato di dare un risvolto *pratico* a questa ricerca, posizionandola nel contesto della *produzione industriale*. In particolare, l'idea è quella di creare un sistema in grado di riconoscere alcune *attività* svolte su *macchinari industriali*, come ad esempio interventi di manutenzione e pulizia, riparazioni, sostituzioni di pezzi e così via.

Come si era già introdotto nella sezione 1.3, manca un dataset specifico per questo tipo di attività e le difficoltà che avrebbe comportato reperirne uno reale di dimensioni adeguate ci ha portato all'idea di sviluppare un nostro *dataset artificiale*, denominato *blender-industrial*.

Il dataset *blender-industrial* contiene video creati *artificialmente* utilizzando la libreria Python messa a disposizione dal software *Blender* (`https://www.blender.org/`), una suite di modellazione e animazione 3D totalmente gratuita ed open source.

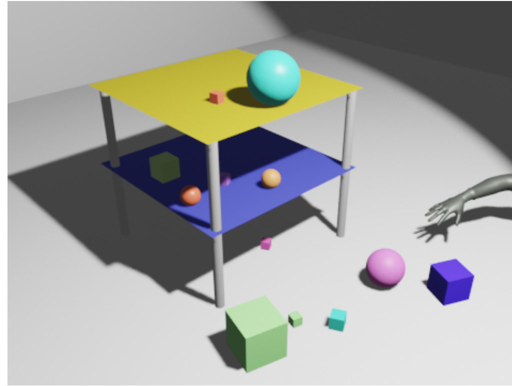
I video così generati approssimano, in maniera chiaramente semplificata, quelle che potrebbero essere alcune registrazioni effettivamente svolte all'interno di un impianto di produzione, quindi pensiamo che un modello pre-addestrato su questi dati possa funzionare con più efficacia all'interno di contesti reali simili.

4.2.1 Ambientazione

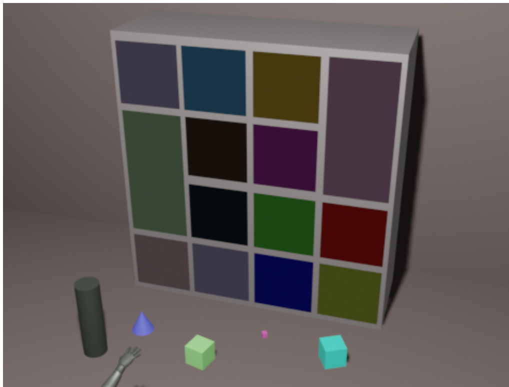
Utilizzando l'interfaccia grafica di Blender, si è modellato un *ambiente tridimensionale* (Figura 4.2) in cui un *braccio meccanico* può interagire con alcuni *oggetti elementari*, come sfere, cubi, cilindri e coni di vari colori e dimensioni.

La prima versione di questo ambiente conteneva un *tavolino* a più piani su cui erano disposti alcuni degli oggetti, poi sostituito nella seconda versione da un *armadietto* con più vani e sportelli di diversi colori (*lockers*).

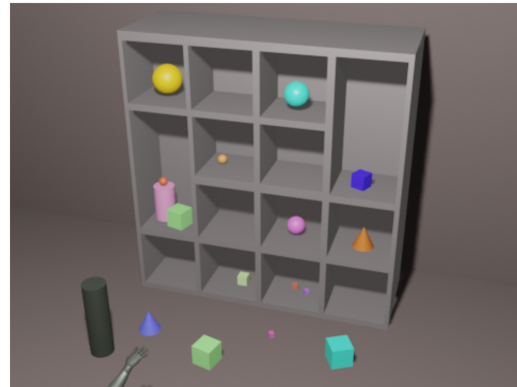
Le immagini sono raccolte da una telecamera *fissa* posta in una *posizione sopraelevata* rispetto ai piani di lavoro, analogamente a quanto verrebbe fatto all'interno di impianti reali.



(a) Prima versione dell'ambiente.



(b) Nuova versione dell'ambiente.



(c) Vista interna degli armadietti.

Figura 4.2: Le due versioni dell'ambiente modellato per lo svolgimento delle attività riprese nei video.

4.2.2 Struttura di un video

In ogni video del dataset è rappresentato lo svolgimento di un'*attività*. Definiamo *attività* una *sequenza* di più *azioni* elementari. Ad esempio, l'attività "cambio di una lampadina" può essere composta dalla seguente sequenza di azioni:

- Posizionamento della scala da lavoro;
- Salita della scala da parte di un operatore;
- Estrazione della vecchia lampadina;

- Inserimento della nuova lampadina;
- Discesa dalla scala;
- Chiusura della scala.

Alcune di queste azioni possono essere effettuate anche durante lo svolgimento di altre attività: ad esempio, le azioni che riguardano l'utilizzo della scala da lavoro possono essere compiute anche per l'apertura di un cassetto difficile da raggiungere.

La scomposizione in azioni più piccole *non-esclusive* rende *impossibile* il riconoscimento di un'attività attraverso una singola immagine (a meno che, ovviamente, l'azione sia caratteristica per tale attività). Diventa quindi necessario per la rete affrontare il problema studiando nel dettaglio le *correlazioni temporali* tra *più fotogrammi* e comprendendo la struttura delle attività tramite *pattern a lungo termine*.

Ciò rende il nostro dataset abbastanza *complesso*. Al contrario, in dataset come Kinetics o UCF101, in cui viene svolta un'unica azione per video, bastano spesso pochi fotogrammi sparsi - se non una singola immagine - per effettuare una classificazione corretta.

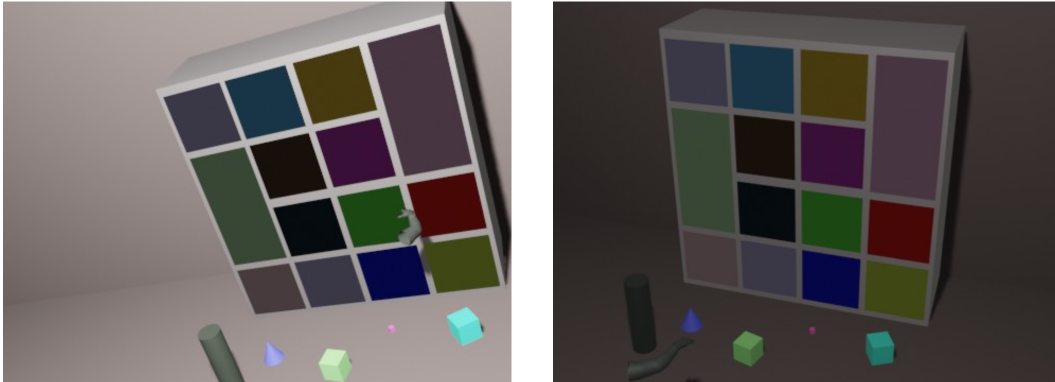
La necessità di cogliere tutte le dipendenze temporali che intercorrono tra le azioni non toglie importanza all'analisi della *dimensione spaziale*. Abbiamo costruito alcune attività *identiche* per quanto riguarda la sequenza di azioni svolte, ma in cui variano gli oggetti coinvolti. L'idea è che due tipi di intervento molto simili possano essere distinti dalla rete attraverso gli strumenti utilizzati dagli operatori.

Questa aggiunta amplia ancora di più lo spazio degli elementi che la rete può considerare per effettuare una predizione, aggiungendo ulteriore sfida per avere risultati più realistici.

Nella creazione dei video sono stati inseriti elementi *randomici* entro certi parametri, in modo da addestrare la rete a *generalizzare* meglio presentandole un'ampia varietà di situazioni.

Gli elementi casuali di ogni video sono:

- Il *colore* e la *rotazione* del *braccio meccanico*, per rappresentare diversi operatori e modalità di utilizzo;
- La *rotazione* della *telecamera* (Figura 4.3a), per rappresentare eventuali interventi che ne possono aver variato la posizione originale;
- L'*illuminazione* della stanza (Figura 4.3b), che indica, ad esempio, diversi momenti della giornata lavorativa.



(a) Fotogramma da un video con rotazione random della telecamera.

(b) Fotogramma da un video in condizioni di scarsa illuminazione.

Figura 4.3: Alcuni fotogrammi che mostrano elementi randomici all'interno dei video del dataset.

Ci sono poi altri elementi casuali relativi alle attività nello specifico. Le dieci attività programmate per il dataset sono le seguenti:

1. *move_sphere_to_other_locker*, un'attività che consiste nel prendere una sfera casuale da un armadietto che ne contiene e spostarla in un altro. Come prima cosa, si aprono entrambi gli sportelli. La sfera viene poi spostata ed inserita in una posizione casuale all'interno del secondo armadietto. Infine, si chiudono entrambi gli sportelli, partendo dal secondo. Il braccio, così come in tutte le altre attività, torna infine alla posizione di partenza (la quale è la stessa per tutti i video).

2. *move_cube_to_other_locker*, analoga all'attività 1, ma l'oggetto trasferito è un cubo.

3. *get_new_object*, in cui il braccio esce all'esterno della scena ripresa dalla telecamera e vi rientra portando con sè un nuovo oggetto casuale tra quelli disponibili. L'oggetto viene posizionato sul pavimento in un punto visibile casuale, dopodiché il braccio torna in posizione di partenza.

4. *put_new_object_in_locker*, che comincia proprio come l'attività 3, ma il nuovo oggetto portato in scena viene poi inserito all'interno di un armadietto casuale. Questo comporta, ovviamente, anche l'azione di apertura e chiusura del relativo sportello.

5. *take_object_out_of_scene*, attività opposta rispetto alla 4: viene scelto un oggetto casuale in un armadietto e lo si porta al di fuori della scena e del campo visivo offerto dalla telecamera.

6. *take_object_drop_and_replace*, la quale consiste nel prendere un oggetto da un armadietto, spostare il braccio meccanico che lo regge al di sopra di una posizione casuale del pavimento visibile e lasciarlo cadere. A questo punto, il braccio meccanico sceglie un altro oggetto tra quelli diposti sul pavimento e lo porta nell'armadietto aperto in cui era disposto il primo oggetto. La porta viene chiusa e il braccio torna in posizione di partenza.

7. *open_three_doors*, in cui il braccio meccanico sceglie casualmente tre armadietti diversi, li apre e torna in posizione di partenza.

8. *swap_from_ground*, molto simile alle attività 1 e 2 in quanto un oggetto casuale viene trasferito da un armadietto ad un altro. Al contrario di queste, però, la prima azione non è quella di aprire entrambi gli armadietti interessati. Si apre solo il primo, si prende l'oggetto e lo si posiziona sul pavimento: solo a questo punto il secondo armadietto viene aperto e vi viene inserito il nuovo oggetto. Concluso il traferimento, uno dei due armadietti viene scelto casualmente per essere chiuso, oppure non ne viene chiuso nessuno.

9. *put_two_objects_on_ground*, in cui, come nell'attività 8, un oggetto viene prelevato da un armadietto e posto sul pavimento. Questa serie di azioni viene ripetuta una seconda volta scegliendo un altro oggetto e alla fine viene chiuso uno dei due armadietti o nessuno.

10. *put_on_top*, dove un oggetto viene preso dall'interno di un armadietto e portato in cima all'armadio, in una posizione casuale. L'oggetto rimane in questa posizione mentre il braccio chiude lo sportello aperto e torna in posizione di partenza.

4.2.3 Implementazione

Si è utilizzato il software Blender nelle *versioni 2.81/82* per impostare l'ambiente tridimensionale, creare e posizionare gli elementi al suo interno. Sono state applicate *texture* diverse agli oggetti e alle porte degli armadietti, si è scelta la posizione della telecamera e il suo campo visivo in modo che fosse inquadrata l'intera scena e si sono presi dei punti di riferimento per controllare i parametri random delle attività, definendo ad esempio l'area sul pavimento utilizzabile per poter appoggiare un oggetto.

Blender è anche in grado di simulare le *interazioni fisiche* tra gli oggetti, che quindi hanno tutti una massa e reagiscono alle collisioni e alla forza di gravità. Il motore fisico fa una distinzione tra oggetti *attivi* e *passivi*: i primi sono direttamente controllati dal motore, mentre i secondi dal sistema di animazione.

Per comprendere meglio questa differenza, si prenda come esempio l'oggetto che forma il pavimento della scena, cioè un piano. Dato che lo spazio di partenza è totalmente vuoto, si può dire che il pavimento "galleggi" nello spazio: tuttavia, si tratta di un oggetto fisico, perché sorregge tutti gli altri elementi della scena e fa sì che non cadano nel vuoto. Questo è un oggetto *passivo*: può mantenere fissa la sua posizione nello spazio ma comunque essere un ostacolo per gli altri oggetti *attivi*.

Tutti gli elementi animati manualmente devono quindi essere oggetti passivi: questi comprendono il braccio meccanico e gli oggetti da esso afferrati. Le sfere e i cubi sul pavimento e negli armadietti sono invece oggetti attivi: in quanto tali le loro animazioni sono delegate al motore fisico di Blender.

Per l'animazione degli oggetti passivi, Blender fornisce una *timeline* in cui è possibile inserire dei *fotogrammi chiave* (*keyframes*), ovvero punti che marcano l'inizio o la fine di *transizioni* relative ad alcuni aspetti degli oggetti. Ad esempio, per spostare il braccio meccanico dal punto A al punto B nell'intervallo di tempo che intercorre tra il frame 40 e il frame 52, è necessario impostare due keyframes sulla posizione del braccio: il primo al frame 40, in cui avremo $location = A$, e il secondo al frame 52 in cui $location = B$. In fase di *render*, Blender calcola la posizione del braccio per ogni fotogramma intermedio in modo che vengano rispettate queste due condizioni, quindi genera l'animazione.

Il numero di transizioni attive in un dato frame è arbitrario, quindi se durante lo spostamento del braccio si vuole far ruotare un altro oggetto o lo stesso braccio, è possibile farlo inserendo i keyframe nel canale adeguato. Questo è il modo in cui, ad esempio, vengono aperti gli sportelli degli armadietti: il braccio trasla nello spazio dalla posizione iniziale della maniglia a quella finale, mentre nello stesso periodo temporale lo sportello ruota di 90° nella dimensione x.

I keyframe sono applicabili anche agli aspetti fisici degli elementi di scena, quindi per lasciar cadere un oggetto basta inserire due keyframe sulla proprietà che indica se l'oggetto è attivo o passivo: quando un oggetto diventa attivo a mezz'aria, la forza di gravità lo fa precipitare inesorabilmente verso il basso.

Blender ha anche un interprete Python built-in che permette di eseguire script ed automatizzare alcuni aspetti della scena. Il software mette a disposizione la libreria Python `bpy` che permette di accedere a tutti gli elementi della scena, crearne di nuovi, eliminarne, modificarne i parametri e inserire fotogrammi chiave nella timeline.

Il seguente snippet di codice mostra come creare un breve video in maniera automatizzata con alcuni parametri random realmente utilizzati nel dataset, tra cui la luminosità della scena, la rotazione della telecamera e la posizione finale di un oggetto che viene spostato.

```
import bpy
from random import choice, random
from math import radians
from mathutils import Vector

#definizione dei possibili livelli di luminosita'
brightness_levels = [x for x in range(2000, 60000, 1000)]

#funzione di utility per ottenere un numero random scalato tra un massimo
    e un minimo
def random_scaled(max, min):
    return min + random() * (max - min)

#funzione per scegliere una tra le luminosita' disponibili
def choose_luminosity(min=2000, max=60000):
    return choice([x for x in brightness_levels if x >= min and x <= max])

#funzione per impostare una rotazione random entro certi parametri fissati
    alla telecamera (con una chance del 20%)
def random_rotate_camera(camera):
    if random() <= 0.2:
        camera_x_max_boundary = radians(69)
        camera_x_min_boundary = radians(56)
        camera_y_max_boundary = radians(20)
        camera_y_min_boundary = radians(-11)
        camera_z_max_boundary = radians(120)
        camera_z_min_boundary = radians(100)
        x_rot = random_scaled(camera_x_max_boundary, camera_x_min_boundary)
        y_rot = random_scaled(camera_y_max_boundary, camera_y_min_boundary)
        z_rot = random_scaled(camera_z_max_boundary, camera_z_min_boundary)
        #effettiva impostazione della rotazione
        camera.rotation_euler = Vector([x_rot, y_rot, z_rot])

#funzione che restituisce una posizione sul pavimento visibile dalla
    telecamera
def select_random_coordinates_on_visible_ground():
    x_min = 2.2875
    x_max = 3.3
    y_min = -3.6
    y_max = 5.00
    return random_scaled(x_max, x_min), random_scaled(y_max, y_min)

#funzione che renderizza la scena con tutti i keyframe e le impostazioni
    settate (da chiamare alla fine)
def render_and_end():
    bpy.ops.render.render(animation=True)

cube = bpy.data.objects["Cube.001"] #accesso ad un oggetto nella scena
sphere = bpy.data.objects["Sphere"] #accesso ad un altro oggetto
```

```
light = bpy.data.lights["Light"]    #accesso al faretto di scena
camera = bpy.data.objects["Camera"] #accesso alla telecamera

#i nomi degli oggetti sono quelli impostati nel file di progetto su Blender

#impostazione random della luminosita' della scena
light.energy = choose_luminosity()

#i due oggetti selezionati vengono resi oggetti "fisici"
cube.rigid_body.enabled = True
sphere.rigid_body.enabled = True
#si rende passivo il cubo perche' verra' controllato tramite i keyframe
cube.rigid_body.kinematic = True
#si rende attiva la sfera perche' verra' controllata dal motore fisico di
    blender
sphere.rigid_body.kinematic = False

#selezione del frame 0 sulla timeline
current_frame = 0
bpy.context.scene.frame_set(current_frame)
#impostazione del primo keyframe per il cubo
cube.keyframe_insert(data_path="location", index=-1)

#dopo 12 frame, il cubo si trova in un altro punto.
current_frame += 12
bpy.context.scene.frame_set(current_frame)
final_x, final_y = select_random_coordinates_on_visible_ground()
#si puo' trattare la posizione di un oggetto come un vettore,
#quindi si impostano le nuove coordinate nel seguente modo:
cube.location[0] = final_x
cube.location[1] = final_y
#impostazione del keyframe finale per il movimento
cube.keyframe_insert(data_path="location", index=-1)

#"baking" della fisica, ovvero calcolo dei keyframe che deve inserire il
    motore fisico. In questo modo, si delega ad esso la gestione di
    eventuali impatti con l'oggetto sphere durante lo spostamento del cubo.
cube.select_set(True)
sphere.select_set(True)
bpy.ops.ptcache.bake_all(bake=True)

#conclusione della scena
render_and_end()
```

Listato 4.1: Breve esempio funzionante che mostra come vengono impostati i principali parametri del video per creare una piccola animazione.

Come si può notare anche solo da questo esempio, il codice per un'attività anche molto semplice può diventare lungo. La creazione dello script reale ha

quindi richiesto la *fattorizzazione estensiva* di diverse parti di codice in comune tra le attività, il che ha sicuramente contribuito a rendere ancora più simili le attività tra loro.

Per la creazione di un video in maniera automatica, Blender deve essere avviato da riga di comando nel seguente modo:

```
./path/to/blender -b /path/to/blender/project/file.blend
-P /path/to/script.py -- (-script-params)
```

dove -b è il parametro che indica a Blender di avviarsi in modalità UI-less, il file di progetto .blend è quello in cui è stato modellato l'ambiente e gli oggetti e con -P si indica lo script da far interpretare dal motore Python built-in di Blender. I parametri dopo il separatore "- -" vengono passati direttamente allo script (ad esempio, il path in cui salvare i video).

Blender permette di selezionare delle impostazioni di render quando avviato con GUI. Nel nostro caso, i video sono stati renderizzati a 12 FPS, producendo come output singoli frame .jpg. I frame renderizzati sono quelli da 0 a 180, quindi l'animazione nel complesso dura 15 secondi.

La fase di rendering è parallelizzabile su più processi ed eventualmente è possibile distribuire il lavoro su macchine multiple e poi aggregare i risultati in un unico dataset, perciò si tratta di un procedimento piuttosto *scalabile*. Il tempo di rendering medio per un singolo video è di circa *70 secondi* e, con le disponibilità hardware personali, è stato possibile avviare fino a 4 processi in parallelo senza cali di performance evidenti.

Per organizzazione, i 180 file jpg del video sono inseriti all'interno di una cartella denominata con il timestamp del momento in cui è stato avviato il render. I video sono organizzati per classe, quindi, ad esempio, il primo frame di un video dell'attività *open_ three_ doors* sarà in:

```
.../open_three_doors/26022020_122601/img_00001.jpg
```

Si è scelto di utilizzare direttamente i frame RGB per velocizzare i test. Inoltre, si sono creati script custom per la generazione delle filelist necessarie al framework mmaction, il quale non supporta questa organizzazione dei video a "3 livelli".

Il nostro dataset potrebbe essere utilizzato anche per il *video captioning*: per ogni video viene infatti generato anche un file CSV (nella stessa cartella dei frame) in cui sono elencate le azioni svolte e gli oggetti in esse coinvolti in un linguaggio *semi-naturale*. La struttura delle frasi utilizzata è mostrata in tabella 4.3 insieme ad alcuni esempi.

who	doesWhat	toWhom	whereAdverb	where	whileDoingWhat	whileToWhom	frameInit	frameEnd
arm	moved	itself	in front of	cube_5			48	60
arm	moved	itself	into	locker_12			60	72
arm	grabbed	cube_5					72	72
arm	moved	itself	out of	locker_12	holding	cube_5	72	84
arm	moved	itself	to	original position			168	180

Tabella 4.3: Struttura delle frasi utilizzata per la descrizione delle attività in linguaggio semi-naturale.

Il codice completo e il file di progetto relativo all'ambiente del nostro dataset sono disponibili sul seguente repository: `https://github.com/volpepe/blender-industrial-dataset`.

4.2.4 Analisi dei risultati

N. Video	Train/Val	N. Att.	Mod.	Metodo	Epochs	Top-1/Top-2
300	21/9	10	RGB	TSN	100	52.22 / 66.67
				I3D	**100**	**66.67 / 82.22**
				SlowOnly	100	38.89 / 58.89
600	42/18	10	RGB	TSN	100	71.11 / 90.00
				I3D	**100**	**90.00 / 97.22**
				SlowOnly	100	82.78 / 94.44
1210	84/37	10	RGB	TSN	100	77.17 / 92.93
				I3D	**100**	**96.20 / 99.73**
				SlowOnly	100	90.76 / 98.37
1810	126/55	10	RGB	TSN	100	88.32 / 98.91
				I3D	**100**	**99.45 / 99.82**
				SlowOnly	100	97.99 / 99.27

Tabella 4.4: Risultati dei test sui modelli con varie versioni del dataset. In **grassetto**, le soluzioni con le accuratezze migliori per ogni versione prodotta. N.Video indica il numero di video contenuti all'interno del dataset, Train/Val il numero di video per ogni classe nel training e nel validation set rispettivamente, N. Att. indica il numero di attività presenti nel dataset, Mod. indica la modality considerata, mentre Top-1/Top-5 le accuratezze percentuali finali. Tutte le versioni del dataset sono bilanciate.

La tabella 4.4 riporta le accuratezze ottenute sul nostro dataset con i modelli già testati nella precedente sezione. Si sono create varie versioni del dataset,

con (circa) 300, 600, 1200 e 1800 video bilanciati tra le classi, in modo da verificare quanto l'addestramento fosse efficace con diverse quantità di dati a disposizione.

La necessità di estrarre informazioni temporali *dense* per la classificazione delle attività trova conferma nelle scarse performance di TSN rispetto agli altri metodi che usano convoluzioni tridimensionali. Come si è già spiegato, TSN segmenta il video in brevi snippet e per il risultato finale fa una media delle classificazioni effettuate sulle singole parti. Tuttavia, gli snippet conterranno solamente *alcune* delle azioni svolte nel video, ed essendo le singole azioni condivise tra multiple attività, risulta veramente *faticoso* per l'algoritmo risalire al label corretto.

Al contrario, le convoluzioni tridimensionali rispondono meglio a questo requisito ed ottengono ottimi risultati anche se viene utilizzata solamente la modality dei frame RGB e solo la Slow pathway nel caso di SlowFast.

I tempi di training e testing di questi metodi sono, tuttavia, *notevolmente* più alti (Tabella 4.5 e 4.6).

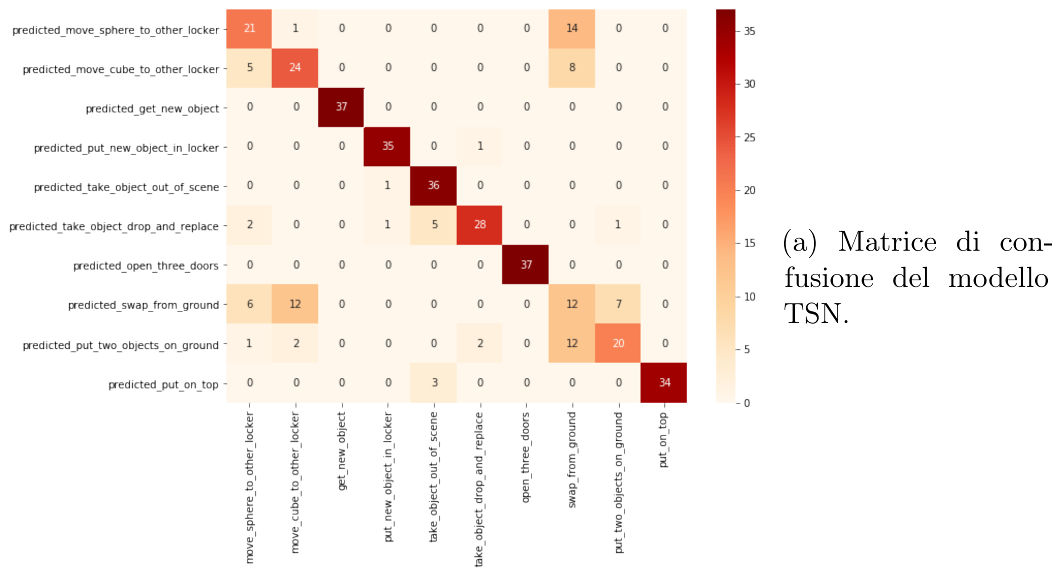
Modello	T. train 1200	T. test 1200	T. test 10	T. eff. 10
TSN	16min, 59s	3min, 15s	19.1s	7.38s
I3D	105min, 3s	17min, 22s	48.1s	33.5s
SlowOnly	59min, 26s	4min, 29s	22.8s	9.99s

Tabella 4.5: Tempi richiesti per il training e il testing sulla versione del dataset con 1200 video per ogni modello. Nella terza e quarta colonna, i tempi di esecuzione e quelli effettivi per il calcolo su un piccolo dataset di 10 video. Si evidenziano i circa 13-16 secondi di overhead per il caricamento della rete e la gestione del parallelismo. Tutti i test sono stati eseguiti caricando i video in parallelo su 2 GPU Titan Xp.

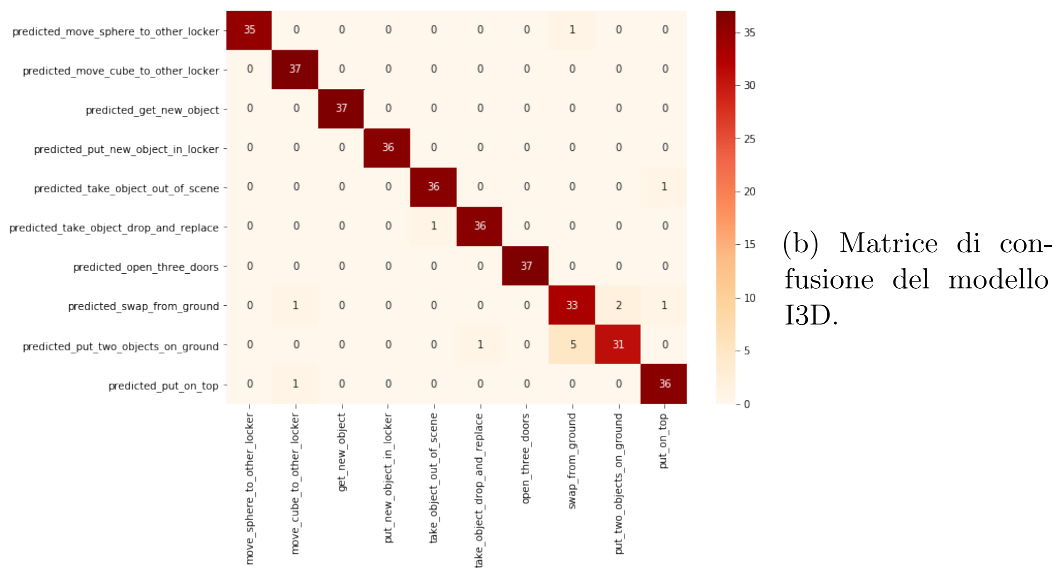
Modello	Esec.	Effett.
TSN	14.4s	3.23s
I3D	20.5s	8.69s
SlowOnly	16.6s	4.47s

Tabella 4.6: Tempi richiesti per l'inferenza su un unico video di 10 secondi su GPU singola.

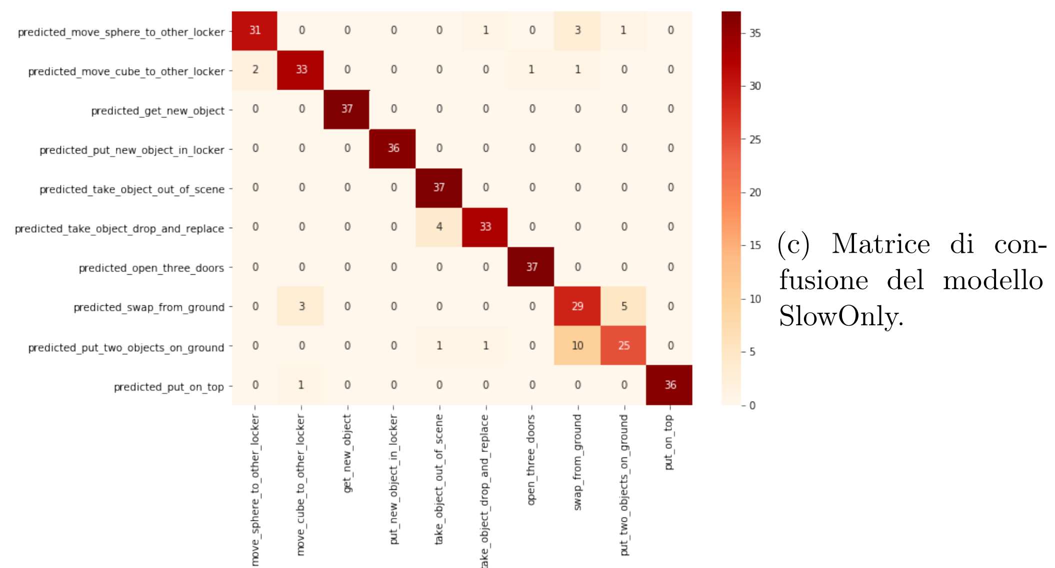
Figura 4.4: Matrici di confusione ottenute testando i modelli sulla versione di blender-industrial con 1210 video (370 video di validation).



(a) Matrice di confusione del modello TSN.



(b) Matrice di confusione del modello I3D.



(c) Matrice di confusione del modello SlowOnly.

La scelta del modello da utilizzare dipende, chiaramente, dalle esigenze del cliente finale. La *vittoria schiacciante* di I3D nell'accuratezza della classificazione viene pagata cara in termini di *complessità computazionale*, mentre SlowOnly rappresenta probabilmente il miglior *compromesso* tra i tre modelli, perché fornisce buoni risultati in tempi accettabili.

In figura 4.4 sono riportate le matrici di confusione della versione di blender-industrial con 1200 video.

La classe *swap_from_ground* è, in generale, la più problematica, tanto che TSN predice in maniera esatta *meno di un terzo dei video* per questa attività.

Tutti i modelli sembrano fare qualche errore nella distinzione tra le attività *swap_from_ground* e *put_two_objects_on_ground*: le azioni compiute in questi video sono, infatti, praticamente identiche tranne nel momento in cui, dopo aver aperto il secondo armadietto, il braccio torna a raccogliere il primo oggetto sul pavimento invece di afferrare quello all'interno.

A volte vengono scambiati anche *swap_from_ground* e i video di *move_cube* e *move_sphere_to_locker*: tutte queste attività sono accomunate dall'apertura di due armadietti e l'estrazione di un oggetto da almeno uno di essi, quindi è possibile che gli algoritmi (specialmente TSN) siano diventati molto capaci a riconoscere queste azioni ma non altrettanto a distinguere le piccole *differenze* che avvengono nel loro svolgimento in sequenza.

Un'altra confusione comune riguarda la classe *take_object_out_of_scene* che viene saltuariamente confusa con *take_object_drop_and_replace*, probabilmente perché spesso l'oggetto che viene fatto cadere rotola fuori dalla scena a causa delle interazioni fisiche con gli altri oggetti.

I video in cui si inserisce un nuovo oggetto in scena e quelli in cui si aprono tre armadietti sono, invece, sempre riconosciuti: probabilmente questo è dovuto alla loro *semplicità* e *originalità* rispetto alle altre attività più confondibili.

Nonostante queste piccole sviste, tutti i modelli svolgono relativamente bene il loro compito, il che ci da fiducia sul fatto che i loro risultati in un contesto reale possano essere buoni.

Conclusioni e sviluppi futuri

Questo lavoro di tesi si è voluto concentrare su un campo vasto e di grande interesse, ma ancora in sviluppo. Le soluzioni testate potrebbero diventare obsolete nel giro di pochi anni, ma ora come ora rappresentano lo *spirito* della ricerca e la loro importanza le renderà probabilmente argomento di discussione per ancora molto tempo.

Il framework mmaction è interessante e viene aggiornato spesso, quindi è probabile che in futuro vengano resi disponibili più modelli e il supporto ad un maggior numero di dataset. Potrebbero inoltre essere pubblicati framework ancora più potenti e di semplice utilizzo, non solo per questo campo, ma per il machine learning in generale.

Il nostro dataset non contiene molte attività e non è ancora comparabile ai dataset reali attualmente pubblicati, ma durante lo sviluppo si sono attuati quanti più accorgimenti possibili per renderlo un progetto facilmente estensibile in futuro. Alcune idee che abbiamo avuto riguardano, ovviamente, l'aggiunta di nuove attività, ma anche l'implementazione dell'esecuzione di *due attività in contemporanea* da due braccia meccaniche che lavorano in porzioni separate dell'ambiente. Si vorrebbe inoltre provare a testare il dataset anche su soluzioni di Video Captioning, come già introdotto alla fine della sezione 4.2.3.

Infine, si vorrebbe capire l'impatto che il pre-addestramento sul nostro dataset può avere su sistemi che analizzano dati reali.

In ogni caso, riteniamo che il lavoro svolto in questa tesi sia un buon compendio dello stato attuale delle tecniche di analisi video e che possa rivelarsi utile per comprendere i passi successivi in questo straordinario campo di ricerca.

Ringraziamenti

Il primo ringraziamento va al relatore di questo lavoro, il Prof. Gianluca Moro, che ha reso possibile tutto ciò ed ha acceso il mio personale interesse nei confronti di questa splendida disciplina.

Ringrazio poi i miei familiari, che mi hanno sempre supportato durante questo percorso universitario e nella vita in generale.

In terzo luogo, voglio ringraziare tutti i miei amici per aver arricchito la mia vita dentro e fuori dall'università, offrendomi esperienze uniche ed indimenticabili e, aggiungerei, aiutandomi - emotivamente o anche in maniere più dirette - alla stesura di questo lavoro. Tra loro, un ringraziamento in particolare va a Mattia Bonoli, che ha messo a disposizione la sua macchina per il rendering di una parte dei video del dataset *blender-industrial* utilizzato per i test nel capitolo 4.

Bibliografia

[1] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedsings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] Chunhui Gu, Chen Sun, Sudheendra Vijayanarasimhan, Caroline Pantofaru, David A. Ross, George Toderici, Yeqing Li, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. *CoRR*, abs/1705.08421, 2017.

[4] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[5] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso A. Poggio, and Thomas Serre. HMDB: A large video database for human motion recognition. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2556–2563, 2011.

[6] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

[7] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, 2016.

[8] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense.

[9] Shaoxiang Chen, Ting Yao, and Yu-Gang Jiang. Deep learning for video captioning: A review. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6283–6290, 2019.

[10] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A large video description dataset for bridging video and language. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5288–5296, 2016.

[11] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[12] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017.

[13] Adam Byerly, Tatiana Kalganova, and Ian Dear. A branching and merging convolutional network with homogeneous filter capsules. *CoRR*, abs/2001.09136, 2020.

[14] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on*

Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Fei-Fei Li. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1725–1732, 2014.

[20] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 568–576, 2014.

[21] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-L^1 optical flow. In *Pattern Recognition, 29th DAGM Symposium, Heidelberg, Germany, September 12-14, 2007, Proceedings*, pages 214–223, 2007.

[22] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4724–4733, 2017.

[23] Yi Zhu, Zhen-Zhong Lan, Shawn D. Newsam, and Alexander G. Hauptmann. Hidden two-stream convolutional networks for action recognition. In *Computer Vision - ACCV 2018 - 14th Asian Conference on Computer Vision, Perth, Australia, December 2-6, 2018, Revised Selected Papers, Part III*, pages 363–378, 2018.

[24] Guojing Cong, Giacomo Domeniconi, Joshua Shapiro, Chih-Chieh Yang, and Barry Chen. Video action recognition with an additional end-to-end trained temporal stream. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7-11, 2019*, pages 51–60, 2019.

[25] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence - video to text. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 4534–4542, 2015.

[26] Shizhe Chen, Jia Chen, Qin Jin, and Alexander G. Hauptmann. Video captioning with guidance of multimodal latent topics. In *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, Mountain View, CA, USA, October 23-27, 2017*, pages 1838–1846, 2017.

[27] Xin Wang, Yuan-Fang Wang, and William Yang Wang. Watch, listen, and describe: Globally and locally aligned cross-modal attentions for video captioning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 795–801, 2018.

[28] Shaoxiang Chen and Yu-Gang Jiang. Motion guided spatial attention for video captioning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 8191–8198, 2019.

[29] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks for action recognition in videos. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(11):2740–2755, 2019.

[30] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *CoRR*, abs/1812.03982, 2018.

[31] François Chollet et al. Keras. `https://keras.io`, 2015.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[34] Dahua Lin Yue Zhao, Yuanjun Xiong. Mmaction. https://github.com/open-mmlab/mmaction, 2019.