Cassetta Davide
Rosati Dalila

# RELAZIONE PROGETTO-ML-SII

## INTRODUZIONE

L'attività di riconoscimento delle azioni prevede l'identificazione di diverse azioni da videoclip (una sequenza di frame 2D), che possono può o meno essere eseguite per l'intera durata del video. Questa disciplina sembra essere un'estensione naturale delle attività di classificazione delle immagini su più fotogrammi, per poi aggregare le previsioni da ciascun fotogramma. Nonostante il grande successo e la precisione piuttosto accurata delle architetture di deep learning nella classificazione delle immagini (ImageNet), i progressi nelle architetture per la classificazione dei video e l'apprendimento della rappresentazione sono più lenti e risultano ancora alquanto imprecisi. Risulta, dunque, tuttora sfidante riuscire ad ottenere risultati soddisfacenti nell'ambito del riconoscimento di categorie e azioni nei video. Verrà ora spiegato nel dettaglio cosa rende così difficile questo compito.

- **Enorme costo computazionale**. Una semplice rete 2D di convoluzione per classificare 101 classi ha solo circa 5 milioni di parametri, mentre la stessa architettura quando viene estesa a una struttura 3D risulta avere circa 33 milioni di parametri. Sono necessari da 3 a 4 giorni per addestrare un *3DConvNet* su *UCF101* e circa due mesi su *Sports-1M*, il che rende la ricerca di un'architettura estesa difficile, sovradimensionata e sensibile al fenomeno dell'overfitting.
- **Cattura di un lungo contesto**. Il riconoscimento delle azioni comporta l'acquisizione di un contesto spazio-temporale attraverso i frame. E' importante sottolineare che le informazioni spaziali acquisite potrebbero essere soggette ad errori a causa del movimento della telecamera e che, anche avere un preciso rilevamento di oggetti spaziali potrebbe non essere sufficiente, in quanto le informazioni acquisite durante il movimento potrebbero evidenziare alcuni dettagli irrilevanti che comprometterebbero l'accuratezza del riconoscimento.
- **Progettazione di architetture di classificazione**. Progettare architetture in grado di catturare informazioni spazio-temporali implica più opzioni non banali e costose da valutare. Ad esempio, alcune possibili scelte da compiere riguardanti le strategie da utilizzare potrebbero essere:
    - Una rete per acquisire informazioni spazio-temporali oppure due reti separate, una per il contesto spaziale e una per quello temporale.
    - Fusione di previsioni su più clip.
    - Training end-to-end oppure separazione delle fasi di estrazione delle caratteristiche da quelle di classificazione.

- **Nessuna standardizzazione dei riferimenti**. I dataset più popolari e di riferimento sono stati per molto tempo *UCF101* e *Sports1M*. La ricerca di

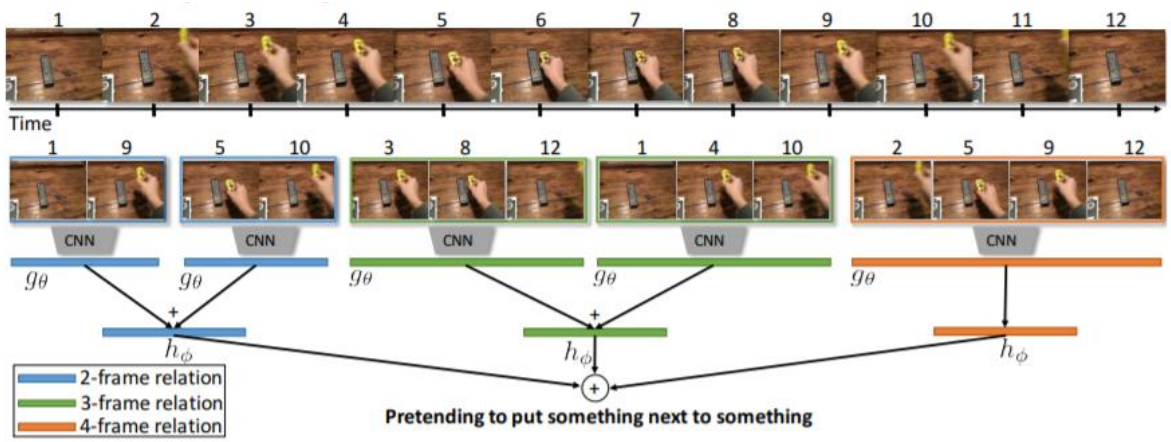
un'architettura ragionevolmente adeguata per *Sports1M* può essere estremamente costosa. Per quanto riguarda *UCF101*, anche se il numero di frame è paragonabile a *ImageNet*, l'alta correlazione spaziale tra i video rende la diversità effettiva significativamente minore durante la fase di training. Inoltre, dato il contesto simile (sport) su entrambi i dataset, la generalizzazione delle architetture volte ad altri compiti è rimasta un problema. La questione è ancora aperta anche per quanto riguarda altre tipologie di dataset e di architetture. Risulta ancora particolarmente oneroso e complesso l'adattamento di un'architettura, progettata per svolgere determinate azioni, ad un dataset fuori dal contesto per essa pensato.

La nascita e lo sviluppo del deep learning ha dunque contribuito a una maggiore attenzione al problema del riconoscimento di categorie e azioni nei video, che un tempo era eccessivamente oneroso. Nonostante ciò, la situazione attuale non risulta ancora del tutto adeguata ad affrontare il complesso compito dell'action recognition.
Il nostro lavoro è basato sullo studio, la manutenzione e la modifica con fini adattativi di un'architettura fornitaci dall'università.
Nel paragrafo dedicato al lavoro svolto, verranno presentate le modalità di analisi dell'architettura, la manutenzione e le implementazioni di importanti sezioni architetturali volte ad estendere il codice a fasi dell'apprendimento, del validation e del testing inizialmente non contemplate. In seguito, verrà esposta la descrizione dettagliata del dataset da noi appositamente creato per testare l'architettura e i risultati successivamente ottenuti.

## STATO DELL'ARTE

I due approcci principali per il riconoscimento di elementi o azioni nelle immagini, si basano su Convolutional Neural Network (CNN) e Recurrent Neural Network (RNN): la prima è molto usata nei task di Computer Vision, soprattutto nel riconoscimento e localizzazione degli oggetti, analizza le immagini una per volta sequenzialmente; la RNN, invece, prende in input una sequenza di immagini che differiscono nel tempo e cerca di stimare le relazioni temporali. Dalla fusione di queste due principali architetture derivano gli approcci maggiormente utilizzati per l'Action Recognition: l'analisi delle relazioni spaziali degli oggetti tra diversi frame e l'analisi delle relazioni temporali. Nel tempo, si è largamente diffusa la Two Stream Network: anziché analizzare un frame alla volta in sequenza, si utilizzano due reti separate che vengono poi ricombinate, una per il contesto spaziale (pre-addestrata) basata su CNN, che analizza i frame singolarmente, e una per quello temporale che prende una sequenza di frame (da 2 a 8) e cerca di estrarre la relazione temporale che c'è tra quei frame, basata invece su un'evoluzione della RNN, la Long-Short Term Memory Unit (LSTM).
Tra le varie architetture attualmente conosciute nel campo del deep learning, volte a riconoscere categorie e/o azioni in dataset di video, l'architettura che abbiamo utilizzato è chiamata Temporal Relation Network (TRN) e si concentra

sull'apprendimento delle dipendenze temporali tra sequenze di frame su diverse scale temporali: si basa sul Temporal Relational Reasoning, che consiste nel predire quello che può accadere, analizzando la situazione corrente; i frame vengono campionati in modo non sequenziale e dati in input a diversi moduli che indagano sulla relazione temporale e spaziale tra essi. Possono essere usate due tipologie di Inception: la Batch Normalization (BNInception), pre-addestrata su ImageNet che offre un buon bilanciamento tra accuracy e efficienza, e la InceptionV3, architettura fornita da Google. Inoltre è possibile usare 2 modalità di input: RGB che dà in pasto all'architettura sequenze di frame RGB o Optical Flow, che descrive il pattern di movimento degli oggetti tra due frame consecutivi, dovuto al movimento della telecamera o dell'oggetto stesso.




LAVORO SVOLTO

Per realizzare il nostro progetto, ci siamo serviti dell'architettura della TRN, disponibile all'indirizzo https://github.com/metalbubble/TRN-pytorch. In seguito, sono state eseguite varie verifiche utilizzando dataset differenti. Dapprima i test sono stati effettuati su macchina remota fornita dall'università, che monta 4 GPU, su dei dataset di prova chiamati Jester e Something-Something-v2 e, successivamente, su Colab, usando un dataset contenente quattro categorie di sport da noi appositamente create. Una prima parte del nostro lavoro è stata dedicata anche alla correzione dei numerosi errori riscontrati nell'architettura fornitaci, probabilmente dovuti all'utilizzo di una vecchia versione di pytorch (0.4), che presenta funzioni ormai deprecate e che entrano in conflitto con le nuove versioni: questi impedivano l'avvio del Training, restituendo degli errori poco chiari ai fini della risoluzione del problema, che hanno richiesto un'analisi accurata del codice per riuscire ad essere risolti. Questa operazione di manutenzione e riscrittura del codice è stata necessaria per poter avviare le fasi di training e di testing. E' inoltre opportuno far presente che l'architettura non contemplava, per la fase di testing, l'utilizzo di un set di video diverso rispetto a quello utilizzato per la fase di validation, in quanto i video utilizzati per la fase di testing erano

gli stessi del validation set; si è rivelato dunque indispensabile implementare del codice per poter includere anche un insieme di sequenze di frame utilizzabili per la fase di testing.

## PRINCIPALI MODIFICHE EFFETTUATE

### Training

Il lavoro svolto per quanto riguarda il training ha previsto un'accurata ispezione del codice, poiché la fase di apprendimento risultava essere colma di errori e non particolarmente adatta ad un cambiamento di dataset. E' stato dunque necessario, dopo aver corretto gli errori, rendere il codice più flessibile e riutilizzabile.

- datasets_video.py: in primo luogo, si è rivelato necessario modificare il percorso root_data da cui l'architettura prende i frame dei video e i file category.txt (file che specifica le categorie dei video), train.txt e val.txt (che contengono gli indici dei video e la categoria assegnata).
- main.py: 1. modifica linea 175 di loss.data[0] in loss.data.item(), che consente di ottenere un valore numerico da un tensor che contiene un singolo valore, perchè nelle versioni successive alla 0.4 di pytorch, ottenere il valore con [0] è deprecato e restituisce un errore. 2. modifica linea 176 di prec1[0] in prec1.item(), linea 177 prec5[0] in prec5.item(), linea 234, 235 e 236 per lo stesso motivo spiegato sopra.

### Testing

La fase di testing, inizialmente non contemplata dall'architettura è stata implementata da noi.

- datasets_video.py: creazione e aggiunta del file test.txt, che permette di prendere gli indici dei video per il testing.
- test_models.py: aggiunta argomento per il file di testing e modifica delle linee 173 e 174 di prec1[0] in prec1.item() e di prec5[0] in pre5.item() per lo stesso motivo spiegato sopra nella sezione training.

## AVVIARE TRAINING, TESTING E TEST SU SINGOLO VIDEO

Prima di avviare il training occorre effettuare alcune operazioni sul dataset:

1. avviare extract_frames.py dando come destinazione la cartella in cui risiedono i video: crea una cartella per ogni video e estrae i frame e li mette nelle cartelle create.
2. avviare process_dataset.py che prende in input i file csv di train, validation, test e le cartelle dei frame e crea in output i file txt relativi, con l'indice del video, il numero dei frame e l'indice della categoria.

Fatto ciò è possibile avviare il training con il comando:

```
python 'path_to_main/main.py' jester RGB \
                --arch BNInception --num_segments 8  \
                --consensus_type TRNmultiscale  --batch-size 64
```

Per il testing invece il comando è:

```
python 'path_to_test_models/test_models.py' jester RGB \
path_to_model/model/TRN_jester_RGB_BNInception_TRNmultiscale_segment8_best.pth.tar'  --arch BNInception --crop_fusion_type TRNmultiscale --test_segments 8
```

Per il test su un singolo video:

```
python 'path_to_test_video/test_video.py' --arch BNInception --dataset jester --weight 'path_to_model/model/TRN_jester_RGB_BNInception_TRNmultiscale_segment8_best.pth.tar' --frame_folder 'path_to_dataset/dataset/sport/frame-video/298'.
```

## CREAZIONE DEL DATASET

Dopo aver provato l'architettura su Jester e Something-v2, ci siamo dedicati alla creazione di un nostro dataset, contenente video riguardanti quattro diverse categorie di sport (soccer, basket, boxing e formula1); il dataset dedica 50 video al training, 12 al validation e 12 al testing per ogni categoria.
Per realizzare il dataset, ci siamo serviti di varie fonti, tra cui Youtube, Instagram e insiemi di database già presenti sul web.
Per preparare la fase di training, è stato svolto un lavoro di ricerca di video coerenti con il tipo di sport che si intendeva rappresentare e, laddove, non è risultato possibile, si è provveduto alla modifica manuale dei video trovati, grazie all'utilizzo di appositi software, al fine di fornire alla rete più inquadrature possibili della stessa categoria sportiva ed evitare eventuali errori dovuti al movimento della telecamera o all'attenzione a dettagli irrilevanti.
Come per il dataset precedente, i risultati derivanti dall'utilizzo dell'architettura non hanno prodotto percentuali alte come invece ci aspettavamo.

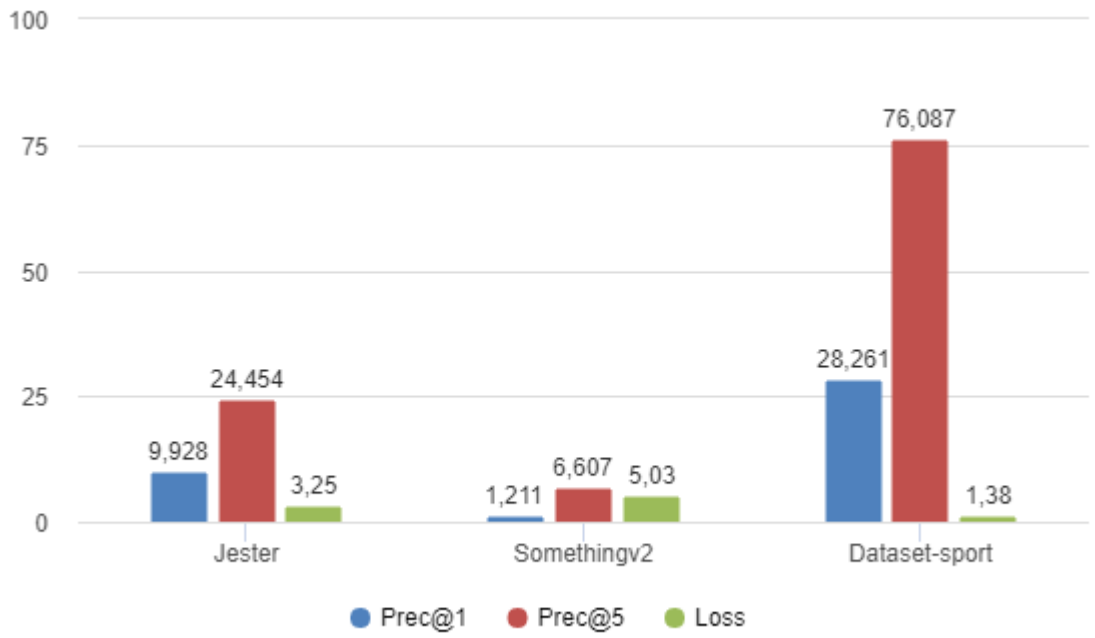
Di seguito si possono osservare dei frame di esempio, utilizzati durante la fase di training, uno per ogni categoria di sport considerata.



RISULTATI

L'esecuzione delle verifiche effettuate sui diversi dataset, utilizzando l'architettura TRN, non ha prodotto i risultati sperati, ovvero le percentuali presentate nel paper descrittivo non coincidono con quelle da noi riscontrate, nonostante avessimo fatto uso degli stessi identici dataset nella fase iniziale del nostro lavoro e dello stesso codice (salvo alcune modifiche che non hanno però intaccato l'architettura esistente). Sono state anche eseguite variazioni degli iperparametri (come batch size, numero di epoche e tipo di Inception) per verificare come questi influenzassero le percentuali di accuracy e loss, ma non hanno prodotto grandi cambiamenti. Ciò è probabilmente dovuto all'impiego di una diversa versione di pytorch o da altri eventuali errori semantici presenti nel codice.
Inoltre, l'architettura è stata sottoposta anche ad estensioni e riduzioni dei dataset utilizzati, ma ciò non ha prodotto risultati significativamente distinti.
In particolare, i risultati fornivano le seguenti percentuali:

| **Dataset** | **Prec@1** | **Prec@5** | **Loss** |
|---|---|---|---|
| Jester | 9.928 | 24.454 | 3.25 |
| Something-v2 | 1.211 | 6.607 | 5.03 |
| Dataset-sport | 28.261 | 76.087 | 1.38 |




CONCLUSIONI

L'attività del riconoscimento delle azioni nei video risulta essere un problema aperto e tuttora ampiamente studiato. Grazie alla nascita e allo sviluppo del deep learning, si sono diffuse sempre più architetture che forniscono percentuali di accuracy più alte che in precedenza e maggiormente efficienti in termini di tempo e memoria.
Le difficoltà ancora esistenti nel riconoscimento delle azioni nei video riguardano soprattutto le relazioni temporali tra frame molto distanti nel tempo e quindi nell'analisi dei video di lunga durata. Da ciò che abbiamo potuto constatare a seguito del lavoro svolto, l'architettura fornita non sembra rispettare le percentuali descritte nel paper informativo. Ciò può essere indice del fatto che l'architettura si serviva di strumenti che sono attualmente ritenuti inefficaci, a causa degli attuali aggiornamenti software che estendono e modernizzano costantemente le librerie presenti. Uno dei nostri compiti è stato quello di sistemare i numerosi errori che presentava l'architettura, che rendevano impossibile avviare le fasi di training, validation e testing ed estenderla in modo tale di consentire l'avvio della stessa su dataset non precedentemente adatti.

BIBLIOGRAFIA


- Paper architettura: https://arxiv.org/pdf/1711.08496.pdf
- Paper stato dell'arte architetture Action Recognition: http://blog.qure.ai/notes/deep-learning-for-videos-action-recognition-review

- ConvNet Architecture Search for Spatiotemporal Feature Learning by Du Tran et al.
- Articolo sul Deep Learning nell'activity recognition: https://machinelearningmastery.com/deep-learning-models-for-human-activity-recognition/