Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Triennale in Ingegneria Informatica

Riconoscimento di azioni in video calcistici usando reti neurali ricorrenti

Candidato
Alberto Baldrati

Relatore
Marco Bertini

Anno Accademico 2018/2019

Indice

Introduzione

Motivazioni

In Italia e nel mondo lo sport, il calcio in particolare, ha una grande importanza sociale e culturale.
A causa di questa importanza e del numero di appassionati, lo sport professionistico è un settore in cui vengono effettuati grandi investimenti e vengono spesi annualmente miliardi di euro, ad esempio i diritti televisivi di serie A per il triennio 2018-2021 sono stati venduti dalla Lega Calcio a **1.41 miliardi** di euro l'anno (fonte calcio&finanza).
Non sorprenderà quindi che una buona percentuale dei soldi che entrano nelle casse dei club calcistici è data prorio dai sopracitati diritti, sempre in serie A nel 2015/2016 tale percentuale era mediamente il **48%** (fonte calcio&finanza).
I broadcaster televisivi, spendendo per tali diritti così tanti soldi e mirando ovviamente a effettuare del profitto, devono avere un gran numero di abbonati. Per raggiungere tale obbiettivo essi offrono servizi di qualità, come ad esempio studi pre-post partita, approfondimenti, ma sopratutto **highlights** che vanno a riassumere i momenti salienti di una partita.
La creazione di quest'ultimi può sembrare un'operazione banale e infatti lo è, tuttavia richiede che una persona annoti manualmente le azioni pericolose

e in particolare i goal.
Questo procedimento dovendo essere ripetuto per tutte le partite rappresenta un costo non indifferente per i suddetti broadcaster, che avrebbero un grande interesse ad automatizzare il procedimento.

Presentazione del lavoro

In questo elaborato di tesi, partendo dal lavoro di Silvio Giancola et al(SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos), verrà descritta una tecnica di automatizzazione del problema citato in precedenza, la quale tuttavia non genera veri e propri highlights, i quali includerebbero oltre ai goal anche azioni pericolose e momenti salienti della partita, ma isola durante l'arco di una partita tre tipi di azione: i **cartellini**, le **sostituzioni** ed i **goal**.
Il lavoro descritto è suddiviso in due parti, la prima parte consiste in un problema di **classificazione** in cui si è proceduto ad addestrare una rete neurale, nella seconda abbiamo usato la medesima rete neurale precedentemente addestrata per isolare le azioni di nostro interesse.
Il Dataset su cui abbiamo lavorato è formato da 500 partite prese dai maggiori campionati europei.

Capitolo 1

Metodo

1.1 Features

1.1.1 Preparazione dei video

Prima di procedere con l'estrazione delle **features** mediante una rete neurale, sono state effettuate sui video le seguenti operazioni:

- Sono stati tagliati in modo che l'inizio del video coincida con l'inizio della partita dopo essere stati sincronizzati mediante OCR
- Dalla risoluzione HD sono stati portati ad una risoluzione di 224 x 224
- Sono stati unificati a 25fps

Tale rappresentazione garantisce video compatibili con l'estrazione di features mediante reti neurali e permette di mantenre un dataset di dimensioni accettabili.

1.1.2 Estrazione delle features

Dopo aver processato i video come descritto sopra si è passati poi alla estrazione delle features.
Per tale operazione è stata usata la rete neurale convoluzionale profonda (ConvNet) **ResNET**, essa data un immagine (nel nostro caso un frame del video) in input, ha come output **2048** features. In particolare è stata usata la variante **ResNet-152** preallenata sul dataset **ImageNet**.
Dato che tale rete neurale è applicata sui singoli frame, essa non mantiene intrinsecamente informazioni temporali, a causa di ciò essa è stata utilizzata per estrarre features ogni 0.5 secondi, preocupandosi successivamente di mantenerle nell'ordine corretto.
Per ridurre le dimensioni della features è stata infine applicata la **Principal Component Analaysis** (**PCA**) che riduce il numero di features per frame da **2048** a **512** mantenendo il **93.9%** della loro varianza.

1.2 Shallow pooling neural network

Nel paper di riferimento vengono testate varie tipologie di pooling, le quali hanno in comune la sigmoide come funzione di attivazione nell'ultimo layer per permettere label multiple in un singolo campione. Le varie tipologie di pooling sono: **mean**, **max**, **CNN**, **SoftDBOW**, **NetFV**, **NetVLAD** e **NetRVLAD**.
Il modello che ottiene i risultati migliori è il **NetVLAD** con 512 cluster, avente nel *fully connected layer* di uscita un *dropout rate* pari a 0.4 che cerca di prevenire l'overfitting.

1.3 GRU-model

Il modello migliore da me sviluppato è una **rete neurale ricorrente** (**RNNs**) basata su **GRU** layer (**Gated Recurrent Unit**). Questi layer sono stati inoltre usati in modo **bidirezionale** in quanto spesso, ciò che avviene dopo un goal, un cartellino o una sostituzione caratterizza l'azione tanto quanto ciò è avvenuto prima, aiutando la rete neurale a classificare in modo corretto l'azione.

Tale modello è stato realizzato con la libreria **keras**, usando come backend **tensoflow**:

```
model = Sequential()
model.add(layers.Bidirectional(layers.GRU(512,
                                          activation='relu',
                                          dropout=0.1,
                                          recurrent_dropout=0.4,
                                          return_sequences=True,
                                          ),
                               input_shape=(None, 512))
          )

model.add(layers.Bidirectional(layers.GRU(256,
                                          activation='relu',
                                          dropout=0.1,
                                          recurrent_dropout=0.4,
                                          return_sequences=True,
                                          )
                               )
          )

model.add(layers.Bidirectional(layers.GRU(128,
                                          activation='relu',
                                          dropout=0.1,
                                          recurrent_dropout=0.4,
                                          )
                               )
          )

model.add(layers.Dense(4,
                       activation='sigmoid')
          )
```

Come si può notare dal modello sono stati utilizzati dei *dropout layer* per cercare di ridurre il problema dell'overfitting, essi sono stati usati sia nella versione standard, sia nella versione apposita per le reti neruali ricorrenti, la quale utilizza la medesima *dropout mask* ad ogni timestamp permettendo di propagare l'errore in modo corretto.
Come otimizzatore è stato utilizzato **RMSProp** con il **learning rate** di default (**lr=0.001**), era tuttavia presente una *callback* che in caso di non miglioramento del modello per più di dieci epoche lo riduceva di un fattore 0.4.
La *loss function* utilizzata, essendo un problema di classificazione binaria multiclasse è la *binary crossentropy*.

Capitolo 2

Esperimenti Classificazione

2.1 Dataset

2.1.1 Descrizione del dataset

Il Dataset su cui sono stati effettuati gli esperimenti descritti in questo elaborato di tesi, è formato da **500 partite**, per un totale di **764 ore** di gioco, prese dai principali campionati europei durante le tre stagioni 2014-2015, 2015-2016 e 2016-2017.
Le partite del dataset sono state divise randomicamente in **300**, **100**, **100** per **training**, **validation** e **testing**, assicurandosi in questo modo una distribuzione similare degli eventi in ogni porzione di dataset.
Dai video delle partite sono state poi estratte le features mediante la rete neurale ResNET con il procedimento descritto in precedenza.

2.1.2 Label

Oltre ai video, il Dataset è formato dalle annotazioni (label), le quali sono state inizialmente ottenute effetuando il *parsing* dei report delle partite

Tabella 2.1: Suddivisone delle partite per lega e anno nel nostro Dataset

	Stagioni			
Lega	14/15	15/16	16/17	**Total**
EN - EPL	6	49	40	**95**
ES - LaLiga	18	36	63	**117**
FR - Ligue 1	1	3	34	**38**
DE - BundesLiga	8	18	27	**53**
IT - Seria A	11	9	76	**96**
EU - CHampions	37	45	19	**101**
Total	**81**	**160**	**259**	**500**

ottenuti sui siti delle varie leghe calcistiche.
Tuttavia la precisione delle annotazioni così ottenute non è abbastanza elevata per i nostri propositi essendo di un minuto, a causa di ciò si sono dovuti riannotare manualmente gli eventi, all'interno del minuto già noto, per ottenere una precisione di un secondo.
Per essere in grado di fare ciò, tuttavia, si sono dovuti definire gli esatti istanti temporali che corrispondono alle azioni di nostro interesse.
Definiamo quindi:

- L'evento **Goal** come l'istante in cui la palla oltrepassa la linea di porta
- L'evento **Cartellino** come l'instante in cui l'arbitro estrae il cartellino giallo
- L'evento **sostituzione** l'istante come in cui il giocatore entra nel terreno di gioco

L'unica eccezione alle definizioni sovracitate sono le sostituzioni avvenute durante l'intervallo, per le quali è impossibile dare una definzione che corrisponda con quanto accade in video.

2.1.3 Suddivisone minuto per minuto

Il nostro dataset formato, come detto sopra, da 500 partite e 764 ore di video, per poter essere elaborato da un rete neurale è stato poi suddiviso in porzioni di **durata di un minuto**.
In questa fase il nostro è un problema di **classificazione**, in cui un singolo campione è formato da 120 raggruppamenti di features ed è annotato con gli eventi occorrenti in quel minuto.
Ovviamente essendo molto più probabile che in un minuto di una partita non accada niente piuttosto che ci sia un'azione da noi ricercata, il nostro dataset è decisamente sbilanciato, esso infatti nel training set contiene:

- **1246** campioni in cui è presete un **cartellino**
- **1558** campioni in cui è prsente una **sostituzione**
- **960** campioni in cui è presente un **goal**
- **23750** campioni di **background**, ovvero in cui non avviene nessuna delle tre azioni sopracitate

È importante notare inoltre come ben **115** campioni abbiano label multiple, per una corretta classificazione è necessario che il modello ammetta la possbilità di classificare questi campioni nel modo corretto.

2.2 Metriche

2.2.1 Mean Average Precision

La metrica usata nel paper di riferimento con cui ci andremo a confrontare è la metrica denominata **Mean Average Precision** (**mAP**). Per il calcolo di tale metrica viene utilizzata la **precision-recall curve**.
Prima di procere oltre è bene introdurre i concetti di **precision** e **recall**:

- La metrica **precison** misura quanto accurate sono le tue predizioni, ovvero la precentuale di previsioni corrette.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.1}$$

- La metrica **recall** misura quanto correttamente riesci a trovare tutti i positivi, può essere pensata quindi come l'abilità di un modello di trovare i punti di interesse di un dataset.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.2}$$

Definiamo quindi **Average-Precision** (**AP**) l'area al di sotto della curva precision-recall.
Infine definiamo **Mean Average-Precision** come la media tra i valori delle AP per ogni evento. Nel nostro caso quindi abbiamo calcolato tre AP: goal, sostituzioni e cartellini e successivamente ne abbiamo fatto la media aritmetica.
Questa metrica sfortunatamente non mi è stato possibile utilizzarla in fase di training (e quindi validazione), questo a causa del fatto che la mAP non è nativamente supportata in Keras, per poterla utilizzare è stato quindi necessario importare tale metrica da Tensorflow.

Le metriche di Tensorflow non supportano tuttavia i dati costruiti mediante funzioni generatici, fatto invece necessario a causa della pesantezza dei nostri dati.
Per questo motivo per effettuare il fine-tuning del nostro modello è stato necessario utilizzare una metrica diversa.

2.2.2 F1

La metrica utilizzata in fase di validazione è stata quindi la **F1**, tale metrica riprende in concetti di Precision e Recall sopra definiti ed ha la seguente espressione:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.3}$$

Nel nostro caso, dato che a noi interessa principlamente indivduare le porzioni di partita in cui accade un evento saliente, tale metrica è stata calcolata solamente sulle **tre** label di nostro interesse **trascurando** quella di background.

2.3 Baseline

I **risultati** ottenuti dalla baseline utilizzando le varie tecniche di pooling precedentemente accennate ed utilizzando le features estratte con ResNET sono elencati di seguito.
Inizialmente sono stati effettuati degli esperimenti con i metodi di pooling senza effettuare alcun tipo di fine-tuning, in particlare sui metodi proposti da *Miech et al* (**SoftDBOW**, **NetFV**, **NetVLAD** e **NetRVLAD**) sono stati usati **k=16** cluster.
Ovviamente all'aumentare dei cluster aumenta la potenza espressiva del sistema, generando con buona prababilità un aumento di prestazioni.

Tabella 2.2: Risultati per i diversi metodi di pooling con $k = 16$

pooling	mAP
Mean Pool.	40.2
Max Pool.	52.4
CNN	53.5
SoftDBOW	48.9
NetFV	64.4
NetRVLAD	**65.9**
NetVLAD	65.2

I risultati sono riassunti nella tabella 2.2.

Tabella 2.3: mAP al variare del numero di cluster k per i metodi di pooling proposti da *Miech et al*

	Metodi di Pooling			
k	SoftBOW	NetFV	NetRVLAD	NetVLAD
16	54.9	63.0	64.4	65.2
32	57.7	64.0	63.8	65.1
64	58.8	64.1	65.3	65.2
128	60.6	64.4	67.0	65.6
256	61.3	63.8	67.7	67.0
512	62.0	62.1	67.4	**67.8**

Nella tabella 2.3 invece si vede come all'aumentare delle dimensioni dei cluster migliorino le prestazioni, tuttavia la complessità computazionale cresce lineramente col numero dei cluster e le prestazionni sembrano stabilizzarsi oltre i 256 cluster a causa dell'overfitting.

Il miglior risultato della baseline, quello quindi con cui ci siamo andati a confrontare, è ottenuto mediante NetVLAD con **k=512** cluster.
Come evidenziato nella tabella 2.3 tale modello sul nostro dataset ha una mAP del **67.8%**.

2.4 I miei risultati

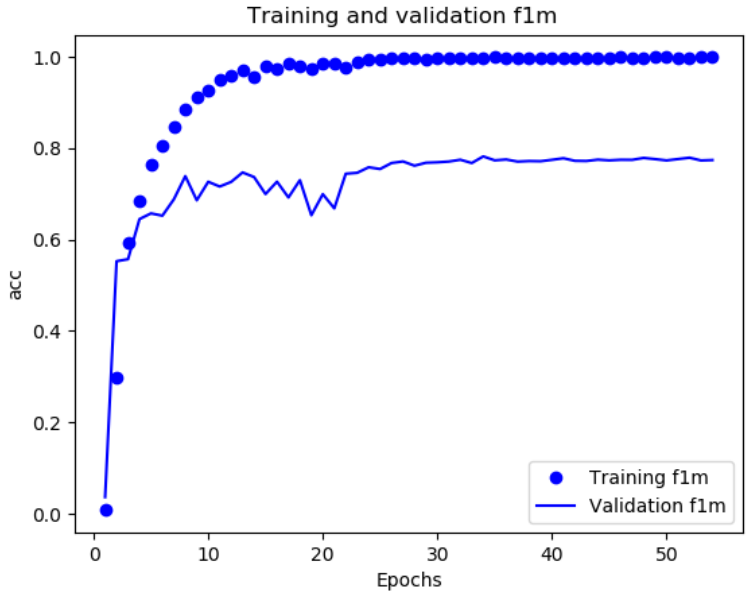
Il modello che è stato utlizzato per cercare di migliorare i risultati ottenuti con la baseline, è stato precedentemente descritto nei dettagli.
Si tratta di una rete neurale ricorrente formata da tre layer GRU bidirezionali con l'aggiunta del layer di ouput.

2.4.1 Training and Validation

Come detto in precedenza a causa dell'incompatibilità tra le metriche di Tensorflow e i dati costruiti mediante generatori in Keras non mi è stato possibile utilizzare la metrica mAP in fase di training del mio modello.
Per questo motivo nel grafico in figura 2.2 viene rappresentato l'andamento della metrica **f1** (denominata **f1m** per sottolinare il fatto che non viene considerata la label di background).
Nella figura 2.1 possiamo vedere come la loss function inizi a crescere dopo circa 10 epoche, facendoci pensare che il modello da li in poi inizi a peggiorare a causa dell'overfitting.
Tuttavia nella figura 2.2 possiamo vedere come la metrica f1 continui a migliorare anche dopo 10 epoche, stabilizzandosi circa alla trentsima epoca e avendo il miglior risultato all'epoca numero 34.
Questa discrepanza è dovuta alla polarizzazione, verso 0 o verso 1, delle probabilità di output del nostro modello.

Figura 2.1: Training and validation loss

Figura 2.2: Training and validation f1m



2.4.2 Test

Per verificare definitivamente la bontà del nostro modello è stato necessario testarlo sulla porzione di dataset riservata a tali propositi. In questa fase la metrica di riferimento torna a essere la mAP.

Tabella 2.4: Risultati sul test set

AP			
Cartellino	Sostituzione	Goal	mAP
77.6	78.7	82.8	**79.7**

Nella tabella 2.4.2 sono illustrati i risultati sul test-set.
Considerando il risultato migliore della baseline contro cui ci siamo confrontati è del **67.8%**, il risultato ottenuto, essendo un miglioramento di oltre il **10%** possiamo ritenerlo soddisfacente.

Capitolo 3

Conclusioni

...

Bibliografia