INTRODUZIONE

Il seguente report descrive il lavoro svolto nell'ambito del progetto per il corso di Sistemi Intelligenti Per Internet e per il corso di Machine Learning. I sorgenti e i notebook del progetto sono disponibili al repository github https://github.com/Meyke/ProgettoSIIML.

Durante il progetto sono stati approfonditi i seguenti argomenti:

- *Recommender System*
- *Face recognition e activity recognition*

In particolare sono stati utilizzati i seguenti tool:

- *PredictionIO*
- *face_recognition*
- *TRN-pytorch*

Il lavoro ha riguardato lo sviluppo di un sistema di raccomandazione di contenuti video sportivi, che filtra (se richiesto) contenuti sulla base di volti riconosciuti e attività riconosciute. Il dataset che si è scelto di utilizzare contiene video riguardanti le olimpiadi di Rio 2016 e a partire da questo è stato addestrato il sistema di raccomandazione e i sistemi di recognition.

Il seguente elaborato è suddiviso nelle seguenti sezioni:

La *parte 1* mostra una panoramica dell'architettura.
La *parte 2* si concentra su come è stata affrontata la parte di raccomandazione e del tool di *PredictionIO.*
La *parte 3* tratta della recognition di volti e attività usando metodi e tool di deeplearning.
La *parte 4* parla dei risultati raggiunti e delle problematiche incontrate
Nella *parte 5* sono tratte le conclusioni.
Nella parte 6 è presente la bibliografia.

1. ARCHITETTURA

L'architettura utilizzata è composta da diversi container docker che comunicano tra loro in rete (una rete docker). I contenitori che fanno deeplearning aprono un proprio jupyter notebook che permette di parlare col sistema di raccomandazione. Tuttavia, occorre tenere in conto che i contenitori che fanno deep learning non possono funzionare se non si ha una gpu NVIDIA con capabilities >= 3.0. Il motivo è descritto nella sezione problematiche. Se non si hanno a disposizione questi requisiti, è possibile utilizzare i tool di deeplearning su colab (opzione preferibile). Sul repository sono stati forniti dei link a proposito.

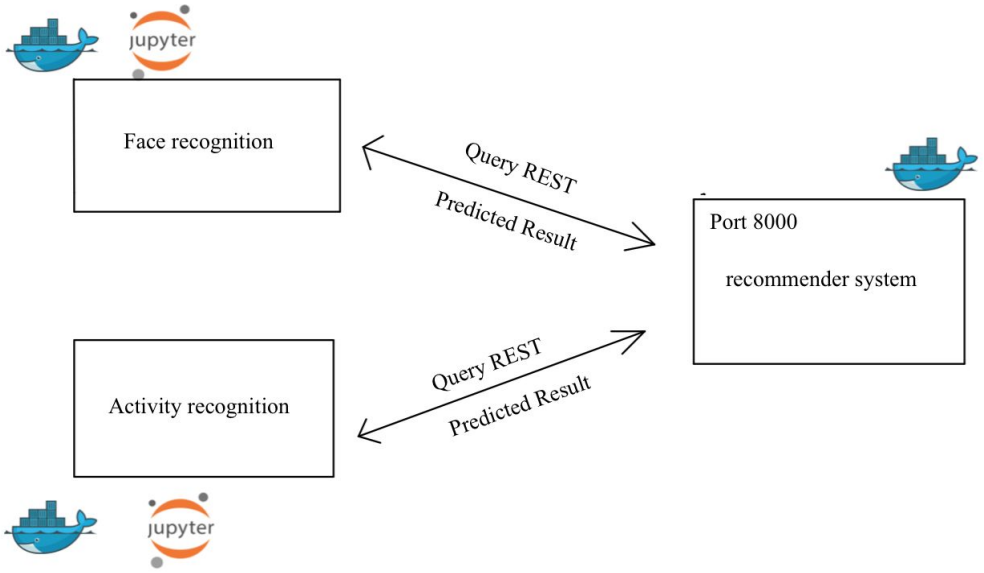

fig.1 Architettura del progetto

2. PREDICTIONIO

PredictionIO è un server machine learning che contiene tutto lo stack software necessario per affrontare problemi in questo ambito. Durante il progetto è stato usato per costruire un motore di raccomandazione. In particolare, si è scelto di utilizzare questo tool in quanto offre numerosi template già preimpostati su cui è possibile lavorare effettuando delle modifiche. La template gallery che offre il sito ha numerosi template utilizzabili in vari campi del machine learning, come raccomandazione, classificazione, regressione, natural language processing, ... Inoltre, tale server fa uso di tecnologie quali Apache Hadoop, Apache Spark, Elasticsearch e Hbase che rendono il tutto scalabile ed efficiente.

PredictionIO consiste nei seguenti componenti:

- *PredictionIO Platform*: lo stack software machine learning che permette di costruire, valutare e fare il deploy di engines con tecniche di ml;
- *Event Server*: fa analisi dei dati provenienti da diverse piattaforme;
- *Template Gallery*: il posto dove scarichi vari engine template in base alle tue necessità.

In generale, **l'Event Server** colleziona dati periodicamente dalle nostre applicazioni. Un PredictionIO engine (che chiamo solo **engine**) poi costruisce un modello predittivo con uno o più algoritmi usando i dati raccolti. Dopo, l'engine viene deployzzato come un web service e quando riceve una query in input dalla nostra applicazione, risponde con risultati predetti in real time.

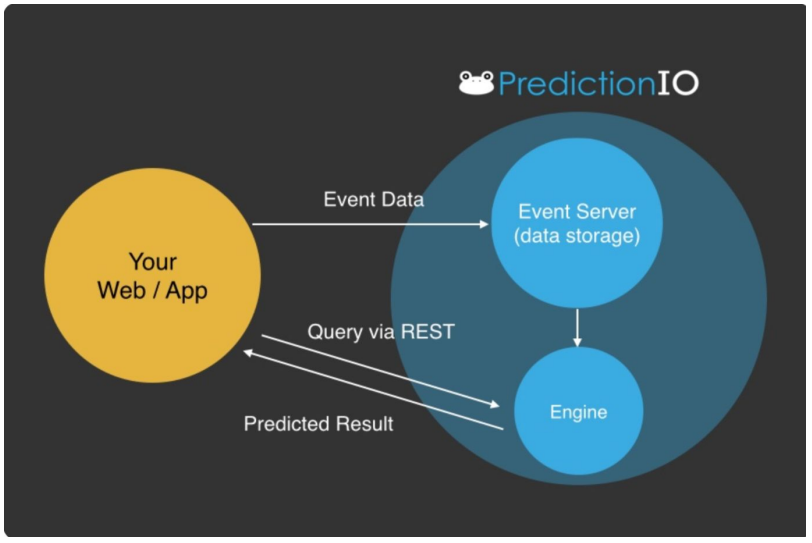

fig.2 Architettura PredictionIO server

Come si nota dalla Fig.2, **l'Event server** colleziona dati dal applicazione, in real time o in batch. Può anche unificare dati provenienti da più sorgenti. Dopo aver collezionato i dati, esso principalmente svolge i seguenti scopi:

- Fornisce i dati raccolti agli engine per il model training ed evaluation;
- Oppure offre una vista unificata per fare analisi dei dati (poco importa).

I dati provenienti da una certa applicazione (sorgente) sono identificati da un univoco *app_name*. Una volta avviato l'Event Server, possiamo inviargli i dati a uno specifico app_name, identificato da un *Access Key*. Possiamo farlo tramite API Rest http o tramite package (SDK) in uno specifico linguaggio di programmazione (es. Java o Python).

L'engine poi legge i dati dall'Event Server. L'**Engine** è il componente che fa predizioni. Esso contiene vari algoritmi di machine learning. Un engine legge dati di training e costruisce modelli predittivi. Poi viene deployzzato come web service in un web server. Un engine deployzzato può eseguire query ricevute dalle applicazioni e fornire risultati in tempo reale

2.1 USARE PREDICTIONIO PER FARE RACCOMANDAZIONE

L'intento del progetto era di utilizzare PredictionIO per poter fare raccomandazione. Tra i template utilizzati è stato preso in considerazione quello di *SimilarProduct*. Tale engine raccomanda prodotti che sono "simili" al prodotto passato nella query, usando un paradigma *Collaborative* (dimmi cosa è popolare tra le persone a me simili). La similarità non è definita dagli attributi degli items o degli users, ma dalle azioni svolte in passato dagli utenti. Di default, esso usa l'evento "view" tale che i prodotti A e B sono considerati simili se la maggior parte che ha visto A ha anche visto B.

Tale template è stato modificato per raccomandare video di sport popolari tra gli utenti anzichè prodotti. Il codice di un engine è scritto in Scala, dato che PredictionIO esegue codice Scala per gli engine, oltre al fatto che tutti gli engine template sono scritti in Scala, forse anche per la semplicità che offre per scrivere Job Spark). Un engine consta di 4 classi principali, chiamati da PredictionIO *componenti DASE*.

Data Source and Data Preparator: *Data Source,* che legge i dati da una sorgente in input e li trasforma nel formato desiderato. Infatti, i dati in input sono semplici JSON in provenienti da chiamate REST, che verranno trasformati in classi Scala (case class). *Data Preparator* preprocessesa i dati e li inoltra all'algoritmo per il training del modello. In realtà tale classe non è stata utilizzata, ma fa solo da passa carte, dato che il preprocessamento è già stato fatto prima dell'invio dei dati al motore di raccomandazione.

Algorithm: Il componente Algorithm implementa gli algoritmi di machine learning. Guardando sia i template offerti e sia la documentazione di Apache Spark MLlib, il modello utilizzato per calcolare la matrice user-item è il *ALS* algorithm (Alternating Least Square). Per la predizione è usata la Cosine Similarity.

Serving: Tale componente è quello che riceve *queries* e ritorna risultati predetti.

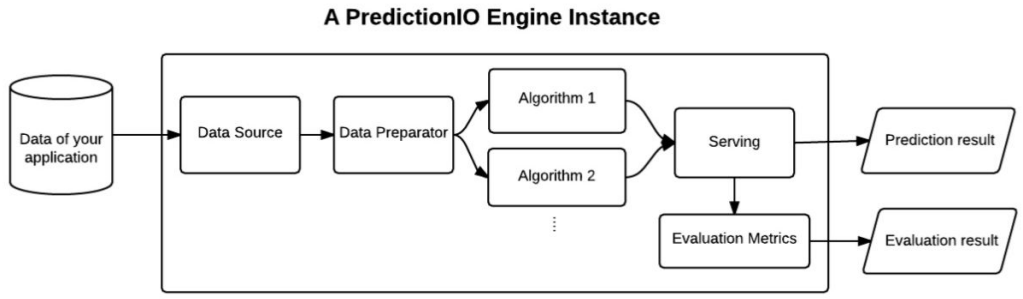

fig.2 Componenti DASE e loro interazioni

2.2 DATI UTILIZZATI PER ADDESTRARE PREDICTIONIO

Per addestrare il modello di predizione utilizzato dall'engine, si è costruito un dataset con le informazioni tratte dalla playlist dei video delle olimpiadi di Rio 2016. I dati sono stati passati in batch all'event server per mezzo del sub comando *import* fornitoci dalla piattaforma. Il dataset è costituito da 3 parti:

- *Dataset di items* che contiene le informazioni circa i video estratti. Tali informazioni sono state estratte usando le API di YOUTUBE;
- *Dataset di users* puramente inventato e contenente solo l'id di ciascun utente;
- *Dataset di users-view* che indica quali video ha visto ciascun utente.

Nella costruzione del dataset degli items sono stati presi in considerazione: id del video, il titolo del video, il titolo del canale, l'id del canaele, l'url del video e vari tag estratti dai metadati del video.

I vari dataset sono stati preprocessati e trasformati in formato json.

2.3 INVIO QUERY ALL'EVENT SERVER

Dopo aver deployzzato l'engine (che sarà in ascolto sulla porta 8000), è possibile inviare query tramite chiamate REST. Il formato delle query sarà anche esso in json. Un esempio di query è la seguente:

```
{ "items": ["EBJWSum8IYQ"],

   "num": 3,

   "tags": ["usain bolt"]}
```

Tale ritorna i primi tre item che più simili all'item identificato dall'id `EBJWSum8IYQ` (secondo la cosine similarity) e in cui compare `usain bolt`, quest'ultimo usato come filtro.

2.4 ESEMPI DI ALCUNE PREDIZIONI

In questa sezione vengono mostrati i risultati ottenuti da alcune predizioni su certe query.

`query lanciata:`

```
{ "items": ["EBJWSum8IYQ"], "num": 3, "tags": ["football"]}
```

`risultati ottenuti:`


```
"itemScores": [
    {
        "item": "8QEdaBLV5is",
        "title": "First Men's Olympic Football gold for Brazil",
        "channelTitle": "Olympic",
        "url": "https://www.youtube.com/watch?v=8QEdaBLV5is",
        "score": 0.9108376766874374
    },
    {
        "item": "EIco9MzmqKU",
        "title": "Neymar scores his first goal in Olympics 2016 with a free kick",
        "channelTitle": "Olympic",
        "url": "https://www.youtube.com/watch?v=EIco9MzmqKU",
        "score": 0.13725112574056603
    },
    {
        "item": "HXBr-IOEOfI",
        "title": "Emmaculate first goal for Zimbabwe",
        "channelTitle": "Olympic",
        "url": "https://www.youtube.com/watch?v=HXBr-IOEOfI",
        "score": 0.020732027643624567
    }
]
```

query lanciata:

```
{ "items": ["pWVyIE30bPs"], "num": 5}
```

risultati ottenuti:


```
{
    "item": "wnzsOYamJqY",
    "title": "Rio Replay: Men's Greco Roman 85kg Gold Medal",
    "channelTitle": "Olympic",
    "url": "https://www.youtube.com/watch?v=wnzsOYamJqY",
    "score": 0.9999482501252718
},
{
    "item": "fsc9-i9LMC4",
    "title": "Van Rijsselberghe wins Windsurfing RS:X gold",
    "channelTitle": "Olympic",
    "url": "https://www.youtube.com/watch?v=fsc9-i9LMC4",
    "score": 0.9999482501252718
}
```


I vari scores indicano quanto l'item della query è simile a tutti gli altri items in base al valore della cosine similarity implementato dal componente DASE Algorithm.

Notare che sono valori puramente indicativi e non vanno presi alla lettera in quanto gli utenti usati per l'addestramento del modello sono fittizi.

3. DEEP LEARNING E TOOL UTILIZZATI

Per quanto riguarda la parte di deep learning, sono stati approfonditi e utilizzati alcuni tool per fare face e activity recognition.
Prima del dominio degli sport si era scelto di lavorare sul dominio dei trailer di video andando a riconoscere facce degli attori presenti nel video ed eventuali azioni importanti. La difficoltà nel utilizzare il tool di activity recognition nel dominio dei trailer di video ha portato poi a deviare verso il dominio di video di sport.
Descriviamo comunque il lavoro che è stato fatto nella prima fase dove si è lavorato nel dominio dei trailer di film.
Nella prima fase si è utilizzato il tool face_recognition di Adam Geitgey per riconoscere facce di attori presenti nel video per poi utilizzare il risultato prodotto dalla parte di deeplearning come input per un sistema di raccomandazione di trailer di film appunto. Il problema maggiore è stato quello di capire come utilizzare il tool per produrre in output direttamente l'etichetta dell'attore riconosciuto. Per fare questo è stato necessario costruirsi un proprio dataset di immagini di attori etichettate con il nome dell'attore.
Nella costruzione del dataset è stata utilizzata una libreria python che permette di scaricare immagini via google specificando i parametri come il nome dell'attore in questione, il numero di immagini da scaricare, il formato, se si preferiscono immagini riutilizzabili, ecc…
La caratteristica fondamentale di questo tool è che inserisce tutte le immagini relative alla ricerca rispetto a un determinato nome direttamente in una directory etichettata con il nome dell'attore. Quindi a partire da un file con una lista di attori e uno script in python che prende in input tutti i nomi della lista di attori, è possibile costruire in modo automatico un dataset di immagini raggruppate per attore.

```
import csv
import os
from google_images_download import google_images_download

# nome del file da cui leggere la lista di attori
file_name = "atleti.csv"

response = google_images_download.googleimagesdownload()

with open(file_name) as file:
    csv_reader = csv.reader(file, delimiter=',')
    for row in csv_reader:
        arguments = {"keywords":row[0],"limit":10,"format":"jpg","size":"medium"}
        response.download(arguments)
```

Successivamente sono state etichettate le singole immagini rinominandole con il nome della directory ed un identificativo numerico crescente.
Il dataset costruito poi si può utilizzare direttamente per l'addestramento, che produrrà i face embedding per ogni immagine presente nel dataset. Nello stesso momento dell'addestramento viene anche costruita una mappa che associa all'identificativo

dell'immagine la corrispondente etichetta, cioè il nome dell'attore che viene estratto direttamente dal nome del file.
Questa mappa che viene salvata in un csv, così come i face embedding, poi viene utilizzata in fase di riconoscimento. Il processo di costruzione del dataset e del successivo addestramento è teoricamente automatizzabile, poiché in fase di addestramento, se vi è un immagine in qualche modo corrotta o che non contiene alcuna faccia da riconoscere, viene restituito un errore che può essere gestito e quindi l'addestramento di una determinata immagine viene scartato e si continua con la successiva. Il dataset potrebbe anche contenere immagini poco chiare o comunque di bassa qualità, ma se questo contiene abbastanza immagini per attore, e visto che la predizione ricadrà sull'immagine più simile presente sul dataset, ci sono ottime probabilità che l'accuratezza non ne risenta. Ovviamente per avere risultati ottimi è consigliata anche una revisione a mano del dataset, andando ad eliminare immagini poco adatte al task in questione.
L'addestramento è svolto dal blocco di codice presente in questo notebook.

```
known_faces = []
count = 0
map = {}

for dirname, dirnames, filenames in os.walk("actors"):
    for subdirname in dirnames:
        subject_path = os.path.join(dirname, subdirname)
        for filename in os.listdir(subject_path):
            try:
                lmm_image = face_recognition.load_image_file(subject_path + "/" +
filename)
                lmm_face_encoding = face_recognition.face_encodings(lmm_image)[0]
                known_faces.append(lmm_face_encoding)
                map[count] =  filename[0:filename.find("_")]
                count = count + 1
                sys.stdout.write('\r'+ "Processamento di " + str(count) + "/" +
str(len(dirnames)*10) + " immagini")
            except:
                print("\nNon è possibile identificare una faccia nel file " + filename)
print("\nProcessamento completato")
```

Navigando e iterando sull'intero dataset, per ogni immagine viene prodotta una rappresentazione vettoriale a 128 dimensioni della faccia riconosciuta nell'immagine.
Il tool, per il calcolo degli embedding, utilizza una rete convolutiva più o meno complicata che ha come strati finali uno strato fully connected con 128 unità seguito da uno strato che fa L2 normalization.
Prima del calcolo degli embedding per ogni immagine viene fatta face detection attraverso l'utilizzo della libreria dlib e poi per ognuna di queste viene calcolato il rispettivo embedding.
Il nostro dataset è stato revisionato in modo da avere solo immagini con una singola faccia, e così da evitare perdite di accuratezza.
Tutto questo viene svolto dal tool attraverso il metodo **face_recognition.face_encodings(image)**.

Il lavoro di riconoscimento degli attori a partire dal video del trailer è presente al seguente notebook.
Nella fase precedente al riconoscimento degli attori, viene svolto un lavoro di tokenizzazione e stopping a partire dal titolo del trailer, in modo da risalire all'id del film su Imdb, poiché il inizialmente il sistema di raccomandazione era stato addestrato per la raccomandazione di film a partire da un dataset preso appunto da Imdb.
Il processamento del video per il riconoscimento sui frame del video è stato svolto attraverso l'utilizzo di OpenCv.
Il tool infatti, iterando sui frame di un video dato un certo frame rate, lavora nel seguente modo:

1. fa detection di tutte le facce presenti nell'immagine
2. per ognuno di queste facce poi viene calcolata una rappresentazione in embedding
3. infine i face embedding calcolati vengono paragonati a quelli già calcolati in fase di addestramento e ai quali è associata un'etichetta. Il tool(attraverso il metodo **face_recognition.compare_faces()**) restituisce una lista degli embedding simili, cioè quelli la cui distanza euclidea è sotto una certa soglia(specificata dal programmatore). Se viene restituita più di una similarità viene poi scelta quella la rappresentazione la cui distanza euclidea è minore. Infine viene restituita l'etichetta associata a quella rappresentazione, che viene estratta dal mapping calcolato in fase di addestramento.

Nella seconda fase del progetto il dominio affrontato è stato quello di video di sport. In questa fase oltre a fare face recognition è stata introdotta anche la componente di activity recognition al fine di riconoscere attraverso l'analisi dei frame lo sport presente nel video.
La parte di deeplearning riconosce appunto atleti presenti nel video e sport praticato e invia tutto ciò al sistema di raccomandazione, che utilizza queste informazioni per filtrare i risultati e raccomandare nuovi video pertinenti all'utente.
In particolare per l'addestramento sia del sistema di raccomandazione che dei tool di deep learning è stato utilizzato un insieme di playlist relativi ai giochi olimpici di Rio 2016.
I video hanno associati un insieme di tag tra cui sono presenti anche gli atleti principali presenti nei vari video e lo sport presente nello specifico video.
A partire dalla lista dei tag presenti nei video, attraverso un join con la lista(https://www.kaggle.com/rio2016/olympic-games#athletes.csv) degli atleti medagliati alle Olimpiadi di Rio 2016, è stata costruita appunto una lista degli atleti presenti nei vari video del dataset preso in considerazione.
Successivamente con lo stesso approccio utilizzato precedentemente nel dominio dei trailer dei film, è stato costruito un dataset di immagini degli atleti principali presenti nei vari video e successivamente è stato fatto l'addestramento per il calcolo dei face embedding relativi a questo dataset,
Per la parte di activity recognition è stato utilizzato il tool TRN-pytorch.

Questo tool è stato sviluppato da ricercatori del MIT con l'obiettivo di creare uno strumento per riconoscere attività a partire da sequenze di immagini, riuscendo a catturare relazioni temporali rispetto a varie scale temporali. La TRN rappresenta un modulo estensibile che può essere utilizzato in modalità plug-and-play con ogni architettura CNN esistente. In particolare qui sono state utilizza reti come BNInception(rete Inception con batch normalization) con congelamento di tutti gli strati che fanno batch normalization, tranne il primo, o reti come la InceptionV3. Le reti utilizzate sono reti preaddestrate su ImageNet.
La TRN, soprattutto nell'approccio "multiscale", ha riportato ottimi risultati su dataset come:

- https://20bn.com/datasets/something-something
- https://20bn.com/datasets/jester
- https://allenai.org/plato/charades/
- http://moments.csail.mit.edu/

In particolare per il nostro progetto è stato utilizzato un modello pre addestrato sul dataset Moments in Time. Il tool permette di riconoscere molte azioni tipiche di molti sport olimpici. Il riconoscimento dello sport viene prodotto sulla base di un mapping azione-sport. L'utilizzo del tool ha portato a scontrarsi con numerose difficoltà, alcune risolte, altra irrisolte. Questa parte di progetto a causa delle difficoltà riscontrate è state sviluppata su notebook su Colaboratory di Google.
Il notebook in questione è presente al seguente url:
https://colab.research.google.com/drive/1Ghs_0S0Fbpmb48_zSTrIpXZF82Uldmnj.
Per le diverse problematiche incontrate consultare l'apposita sezione.

L'utilizzo di questo tool per riconoscere sport a partire da un video ha prodotto ottimi risultati.
Consultare anche in questo caso l'apposita sezione per la descrizione dei risultati raggiunti.

4. RISULTATI RAGGIUNTI

Non sono stati effettuati test specifici per la parte di face recognition. Le prove fatte per la parte di face recognition, soprattutto per il dominio degli attori, sembra portare ad ottimi risultati.
Per la parte di face recognition nel dominio dello sport, l'accuratezza è minore, poichè ci si è soffermati maggiormente sulla parte di activity recognition e si è trascurata la buona costruzione di un dataset.
Per fare un test più approfondito e affidabile bisognerebbe scegliere un dataset di trailer, estrarne attraverso metadati o l'utilizzo di qualche libreria apposita (come quella di Imdb) la lista degli attori dei vari film del dataset, costruirsi un buon dataset di immagini su quegli attori (come già detto con gli script utilizzati questo processo è facilmente automatizzabile, ma per una buona accuratezza una revisione del dataset va effettuata), utilizzare il tool addestrato per fare le predizioni sul dataset di trailer scelto e verificare l'accuratezza dei risultati confrontandoli con la lista di attori realmente presenti in quel film. Per una buona misura di accuratezza un'idea potrebbe essere quella di contare le previsioni esatte e quelle errate, e normalizzare il risultato rispetto al numero di film per esempio. Per dei semplici test è possibile utilizzare il codice della parte di face recognition sugli attori presente nel repository, dando in pasto al codice il trailer che si preferisce cambiando l'url nel codice. La lista degli attori riconoscibili è presente anch'essa all'interno del repository.
Sono stati effettuati dei test per verificare le prestazioni del tool di activity recognition relativamente al riconoscimento di sport. In particolare il tool è stato utilizzato e testato per riconoscere alcuni sport individuali olimpici.
Il tool, attraverso il mapping azione-sport effettuato, è in grado di riconoscere e predire con ottime prestazioni video relativi ai seguenti sport olimpici individuali:

- Weightlifting
- Wrestling
- Tennis
- Table tennis
- Swimming
- Sailing
- Equestrian
- Boxe
- Fencing
- Cycling
- Golf

- Diving
- Athletics
- Rowing
- Taekwondo

Il test è stato fatto prendendo in considerazione un dataset di circa 20 video(ovviamente diversi da quelli utilizzati per la creazione del mapping azione-sport) per ciascuno degli sport precedentemente elencati. Il dataset è stato costruito a partire da alcune playlist di Youtube.
Il test è effettuato nel seguente notebook ed è stato sviluppato su colab, anche in questo caso per la necessità di avere uno stack cuda su cui far girare il tool di activity recognition.
Il dataset in questione è presente al seguente file csv nel repository del progetto. Il dataset è un elenco di link di 275 video, ognuno dei quali è associato allo sport presente nel video.
Il codice nel notebook iterando sull'intero dataset predice lo sport presente nel video e confronta il risultato con lo sport realmente presente.
Il risultato del test in particolare ha restituito i seguenti risultati. Di 275 video:

- in 240 lo sport è stato riconosciuto correttamente
- in 11 lo sport è stato riconosciuto in modo errato
- in 11 non è stato riconosciuto alcun sport
- i rimanenti 13 video hanno prodotto eccezioni del tipo "la visione del video non è possibile nel vostro paese", che ne hanno impedito l'analisi.

Quindi scartando i 13 video non letti, e quindi sulla base di 262 video analizzati, il tool relativamente ai 15 sport riconoscibili, ha prodotto un'accuratezza del 91,6%.

4.1 PROBLEMATICHE INCONTRATE

Le maggiori problematiche sono state incontrate durante l'utilizzo e il test del tool di activity recognition. Inizialmente è stato riscontrato il seguente errore: https://github.com/metalbubble/TRN-pytorch/issues/31. Il problema è stato risolto modificando una specifica riga all'interno del file https://github.com/yjxiong/tensorflow-model-zoo.torch/blob/e31e0b7aa451e2c12c0107e616953a03d8cd0d47/bninception/pytorch_load.py.
L'addestramento e l'utilizzo del tool di activity recognition richiedono necessariamente una macchina con gpu NVIDIA e stack cuda. In particolare è richiesta una gpu con cuda capabilities di almeno 3.0.
Ed è per questo motivo che la parte di activity recognition è stata sviluppata su colab. In ogni caso nel repository, nella directory di sports recognition, è stato inserito sia un notebook col codice per fare activity recognition per gli sport menzionati nella sezione dei risultati raggiunti che un dockerfile con stack cuda nel caso si abbia a disposizione una macchina con gpu e le giuste capabilities, per eseguire il tool sull'apposito contenitore.

Il funzionamento del tool è stato testato con successo sulla macchina tesla che ci è stata messa a disposizione. Sfruttando le potenzialità della macchina si è tentato anche di

addestrare il tool per riconoscere direttamente lo sport, e non l'azione nello specifico.
Abbiamo inizialmente tentato l'addestramento su un piccolo dataset costruito da noi, ma senza buoni risultati.
Allora abbiamo poi scelto di provare ad addestrare il tool sul seguente dataset di sport: http://www.cse.msu.edu/~liuxm/sportsVideo. Questo dataset contiene 4200 video di 30 sport diversi. È stata effettuata una fase di preprocessamento del dataset, in modo da renderlo pronto per l'addestramento. Questa fase ha richiesto di creare uno script che utilizza Ffmpeg per estrarre i frame dai vari video del dataset(che sono in mp4), script che creassero i csv con la corrispondenza tra directory con i frame di un singolo video e l'etichetta sportiva associata al video; si praticamente messo il dataset nella stessa forma dei dataset utilizzati dai creatori del tool. Il dataset è stato diviso in training, validation e test set con percentuali del 80,10,10.
Sono stati tentati vari tentativi di addestramento provando a cambiare diversi parametri come numero di epoche, batch size, learning rate, momentum, dropout, decay, ma ottenendo sempre risultati con accuratezza sul test set minori al 25%.
Nell'addestramento della TRNmultiscale si è cercato di addestrare la rete cercando di individuare relazioni fino a 30 frame ma senza grossi miglioramenti.
Gli addestramenti sono abbastanza lunghi e pesanti(a volte gran parte della memoria delle quattro gpu veniva occupata durante l'addestramento), soprattutto se si vogliono individuare relazioni su diverse scale temporali, anche a lungo termine.
La macchina spesso è stata occupata in questo periodo impedendo addestramenti molto lunghi che potevano magari portare a dei risultati migliori.

5. CONCLUSIONI

In conclusione durante questo progetto è stato fatto uso di tutta una serie di tool di sistemi di raccomandazione e deeplearning. Abbiamo approfondito la teoria alla base di questi strumenti e ne abbiamo poi fatto uso testando il funzionamento in scenari reali.
Sono stati affrontati problemi come la necessità di elaborazione dei dati prima dell'utilizzo, l'utilizzo di reti, preaddestrate e non, molto famose nell'ambito del deeplearnig, l'utilizzo di architetture e macchine gpu(con tutte le problematiche del caso) di tipico utilizzo in questi ambiti e l'utilizzo di molti dataset reali presenti allo stato dell'arte.
Il lavoro svolto è presente al seguente repository:
https://github.com/Meyke/ProgettoSIIML.git
Inoltre i seguenti notebook sviluppati su Colab fanno parte del lavoro svolto:

- https://colab.research.google.com/drive/1Ghs_0S0Fbpmb48_zSTrIpXZF82Uldmnj
- https://colab.research.google.com/drive/112Gu3yd4BgUw4ebCQ9woFGUxy_YpsIIl

6. BIBLIOGRAFIA

PredictionIO: https://predictionio.apache.org/

Template Similar Product PredictionIO:
https://predictionio.apache.org/templates/similarproduct/quickstart/

TRN-pytorch: https://github.com/metalbubble/TRN-pytorch

Face recognition di Adam Geitey: https://github.com/ageitgey/face_recognition

Librerie di python utilizzate durante il progetto:

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
- https://imdbpy.sourceforge.io/
- https://pythonhosted.org/Pafy/
- https://www.nltk.org/
- https://zulko.github.io/moviepy/

Dataset utilizzati:

- http://www.cse.msu.edu/~liuxm/sportsVideo
- https://www.youtube.com/user/olympic/playlists?view=50&sort=dd&shelf_id=141
- http://moments.csail.mit.edu/