POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Gestionale

Tesi di Laurea Magistrale

Classificazione di sequenze di inquadrature cinematografiche tramite reti LSTM

Relatore
Tania Cerquitelli

Candidato
Letizia Demarchi

Correlatore
Bartolomeo Vacchetti

Anno Accademico 2020/2021

Sommario

Abstract

Lo studio della presente tesi si concentra su una specifica categoria di algoritmi di apprendimento supervisionato, ovvero gli algoritmi di Classificazione, che sono stati impiegati per l'analisi di sequenze di inquadrature cinematografiche. In seguito ad un iniziale approfondimento sui temi dell'Intelligenza Artificiale e del Deep Learning, ci si è concentrati sull'analisi delle reti neurali LSTM, le quali risultano particolarmente adatte per la classificazione di un modello di questo tipo. Successivamente, si è passati alla creazione del Dataset su cui svolgere le successive analisi: esso è stato caratterizzato da diverse informazioni tra cui la sequenza di inquadrature della clip in considerazione ed il relativo genere cinematografico. L'obiettivo della tesi è, infatti, quello di comprendere grazie ad un approccio supervised se, dopo aver correttamente addestrato un modello di rete neurale sulla base della sequenza di inquadrature, esso è in grado di definire correttamente il genere.

Introduzione

Il presente lavoro di tesi si pone come obiettivo quello di analizzare tramite algoritmi di Classificazione, le sequenze di inquadrature cinematografiche e comprenderne le eventuali correlazioni del genere.

Lo studio è articolato in diversi capitoli. Nel primo capitolo viene fatta un'introduzione generale sul Deep Learning, partendo dal concetto di Intelligenza Artificiale.

Nel secondo capitolo viene approfondito lo studio delle reti RNN poiché in quest'analisi verranno utilizzate le reti LSTM. Inoltre vengono comparate le reti LSTM con le reti CNN e ne vengono evidenziate le differenze e i diversi campi in cui vengono utilizzate.

Nel terzo capitolo ci si concentra sulla Sequence Analysis e le metodologie principali con cui tendenzialmente viene trattata.

Il quarto capitolo rappresenta il fulcro dell'analisi in quanto in esso viene illustrato il modello di Classificazione e vengono analizzati i risultati ottenuti. In particolare, vengono presentati diverse tipologie di Cross Validation, utilizzate per cercare di migliorare le performance del modello stesso.

Nel quinto capitolo viene proposto un secondo modello che invece ha come scopo quello di presentare due tipologie di analisi con le reti LSTM per comprendere come poter migliorare il modello inserendo il Meccanismo di Attenzione nell'analisi.

In ultimo vengono presentate le conclusioni del lavoro di tesi, evidenziandone i limiti ed i possibili sviluppi futuri.

Capitolo 1: Deep Learning

1.1 Introduzione al Deep Learning

Intelligenza Artificiale, Machine Learning e Deep Learning sono tre termini che spesso si utilizzano in modo quasi identico ma che hanno tre significati molto diversi. Immaginiamo di vederli in una serie di insiemi come rappresentato nella *figura 1*: l'insieme più esterno è quello dell'Intelligenza Artificiale che a sua volta contiene il Machine Learning, mentre quello più interno è rappresentato dal Deep Learning.

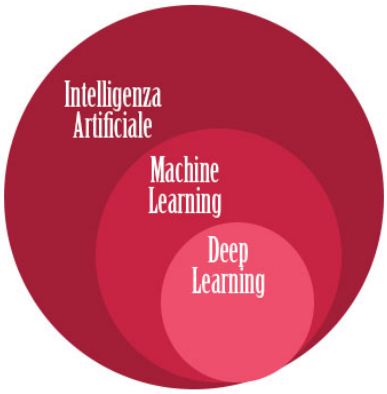

Figura 1 - IA, ML, DL

L'Intelligenza Artificiale è l'obiettivo da raggiungere; la volontà infatti, è quella di creare algoritmi di intelligenza artificiale in grado di pensare e risolvere problemi esattamente come farebbe un essere umano.

Il Machine Learaning, invece, ha come scopo quello di stravolgere il normale modo con cui si è abituati a programmare. Infatti, prima dell'avvento dei classificatori, venivano creati degli algoritmi completi in cui il comportamento del sistema veniva descritto riga per riga. Ora, l'idea è quella di ottenere un sistema che, basandosi su un grande set di dati ed una serie di algoritmi di classificazione, senza scrivere tutto l'algoritmo sia in grado di prendere delle decisioni sulla base dei dati che gli vengono messi a disposizione.

Nell'insieme più piccolo, invece, troviamo il Deep Learning. Esso è una tecnica di Machine Learning che si basa su una parte di conoscenza che già abbiamo, ovvero quella delle reti neurali che sono invece definibili come dei software in grado di mimare, sotto alcuni punti di vista, il funzionamento dei neuroni. Alla base del Deep Learning abbiamo la profondità delle reti neurali, cioè il numero dei livelli di quest'ultima che risulta essere notevolmente elevato. Il concetto su cui si basa l'apprendimento profondo è molto interessante: è la macchina stessa a determinare i classificatori, sulla base di una quantità di dati spesso molto più vasta di quella usata dal normale Machine Learning. Essi dunque, non sono scelti a priori dai ricercatori, ma sembrano avere un impatto estremamente superiore da un punto di vista di risultati che vogliamo conseguire.

1.2 Intelligenza Artificiale

L'Intelligenza Artificiale (A.I. Artificial Intelligence), per come viene concepita oggi, nasce parallelamente all'avvento dei computer e la sua data di nascita viene fissata come il 1956. In quell'anno, a Dartmouth nel New Hampshire, si tenne un convegno dedicato allo sviluppo di macchine intelligenti. L'iniziativa, proposta da un gruppo di ricercatori guidati da John McCarthy, aveva come obiettivo quello di creare in pochi mesi una macchina capace di simulare l'apprendimento e l'intelligenza umana. Il convegno vide la partecipazione dei più importanti studiosi del tempo di quella che sarebbe stata solo successivamente definita Intelligenza Artificiale, ma che allora era consuetudine chiamare Sistema Intelligente.

Gli anni successivi al 1956 furono anni di grande fermento intellettuale e sperimentale: università e aziende informatiche, indirizzarono le loro ricerche verso lo sviluppo di nuovi programmi e software che si ponevano come obiettivo quello di pensare e agire come gli esseri umani. Nacquero così programmi in grado di dimostrare teoremi sempre più complessi e, soprattutto, nel 1958 venne creato il Lisp, ovvero il primo linguaggio di programmazione che per oltre trent'anni fu la base dei software di Intelligenza Artificiale. [1] [3]

Durante la seconda metà degli anni sessanta, divenne sempre più evidente che quanto realizzato fino ad allora in questo ambito non fosse più sufficiente viste le nuove necessità. Le esigenze erano cambiate e lo scopo non era più solo quello di risolvere teoremi matematici più o meno complessi, ma ragionare e prendere delle decisioni successivamente all'analisi di differenti possibilità. Questo tipo di obiettivo prevedeva, prima di poter essere risolto, la soluzione di un altro step: la realizzazione di percorsi semantici per le macchine. Era necessario dunque, un linguaggio che permettesse di programmare le diverse possibilità previste da un ragionamento, semplice o complesso che fosse. Come spesso accade però, il passaggio da uno step ad un altro fu tutt'altro che semplice: la ricerca in questo settore subì un brusco rallentamento, soprattutto perché a causa dell'iniziale scarsità di risultati, tutti i finanziamenti per questo tipo di ricerca furono drasticamente ridotti. [1]

A partire dagli anni '70 furono sviluppati diversi sistemi esperti, ovvero programmi in grado di affrontare un problema specifico simulando le capacità di un esperto in quel particolare ambito; in altre parole, essi erano capaci di conservare, organizzare e proporre

all'utente la conoscenza di un preciso dominio ma, soprattutto, dedurre da determinate azioni o domande anche la conoscenza utile seppur non espressamente richiesta. Nel 1976 infatti, fu creato MYCIN, un sistema esperto in grado di fare diagnosi per malattie ematiche. È negli anni '80 che l'Intelligenza Artificiale esce dall'ambito accademico ed entra nel mondo industriale. Nel 1982 fu introdotto R1, il primo sistema esperto utilizzato in ambito commerciale che permetteva di configurare gli ordini per nuovi computer. [1] [2]

Da quel momento ai giorni nostri le applicazioni basate sull'intelligenza artificiale si sono moltiplicate. Questo successo dell'A.I. è dovuto all'evolversi delle capacità computazionali e allo sviluppo di una serie di tecnologie abilitanti, tra le quali troviamo Big Data e cloud storage.

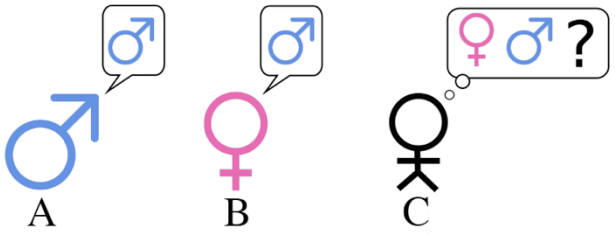

Figura 2 - Test di Turing primo step

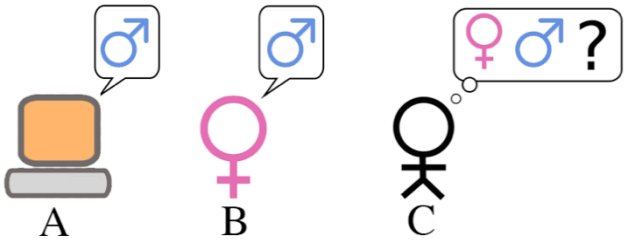

Figura 3 - Test di Turing secondo step

Inizialmente, per riuscire a dare una definizione di Intelligenza Artificiale, veniva utilizzato il test di Turing. Nel 1950 infatti, Alan Turing propose il cosiddetto "gioco dell'imitazione", ovvero un paradigma per stabilire se una macchina è "intelligente". Egli, come si può vedere nella *figura 2,* suggeriva di porre un osservatore di fronte a due telescriventi. Una delle due è comandata da un uomo, l'altra da una donna. L'osservatore, che non sa a quale terminale corrisponda l'uomo e a quale la donna, lo può accertare ponendo loro qualunque tipo di domanda. Uno dei due interlocutori deve rispondere con sincerità, l'altro invece deve fingere d'essere dell'altro sesso. Successivamente, come mostrato nella *figura 3*, all'interlocutore "bugiardo" si sostituisce un calcolatore programmato in modo da fingere di essere una persona umana. Quando sarà pari il numero d'errori prodotto nei tentativi di identificare il calcolatore a quello verificatosi nel caso dell'identificazione dell'interlocutore bugiardo, allora si potrà dire che il calcolatore è "intelligente". [3]

Dal 1991 si svolge ogni anno il Loebner Prize, una competizione che si basa proprio su questo test per valutare le capacità delle A.I. e poterle definire quindi veramente intelligenti. In questa sfida, alcuni giudici intrattengono una conversazione scritta con dei bot per valutare quanto questi ultimi presentino dei comportamenti simili a quelli degli esseri umani. [3]

Poiché ad oggi l'A.I. ha superato il test di Turing solamente tre volte, esso non viene più considerato come un vero e proprio discriminante ed è stata introdotta una definizione più "blanda". L'Intelligenza Artificiale viene dunque vista come un insieme eterogeneo di tecniche e metodi volti a costruire sistemi artificiali dotati di capacità cognitive che siano quindi capaci di riconoscere, classificare, ragionare, diagnosticare e anche agire, o che siano dotati almeno di alcune di queste proprietà. [4]

1.3 Machine Learning

Il Machine Learning, anche definito apprendimento automatico, come già introdotto precedentemente, è un metodo di analisi dati che automatizza la costruzione di modelli analitici. È una branca dell'Intelligenza Artificiale e si basa sull'idea che i sistemi possano imparare dai dati, identificare modelli autonomamente e prendere decisioni con un intervento umano ridotto al minimo. Gli algoritmi di Machine Learning migliorano le loro prestazioni in modo "adattivo" mano a mano che gli esempi da cui apprendere aumentano.

Nel 1959 Arthur Lee Samuel, scienziato americano pioniere nel campo dell'Intelligenza Artificiale, coniò per primo il termine. Ad oggi, però, la definizione più accreditata dalla comunità scientifica è quella fornita da un altro americano, Tom Michael Mitchell, direttore del dipartimento Machine Learning della Carnegie Mellon University. Egli afferma che "*si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E*". [5]

Il Machine Learning permette ai computer di imparare dall'esperienza. C'è apprendimento quando le prestazioni del programma migliorano dopo lo svolgimento di un compito o il completamento di un'azione, sia in modo corretto che in modo errato, partendo proprio dall'assunto che anche per l'uomo vale il principio "sbagliando si impara". [6] Questi meccanismi permettono a una macchina intelligente di migliorare le proprie capacità e

prestazioni nel tempo. L'aspetto più importante dell'apprendimento automatico infatti, è la ripetitività: più i modelli hanno dati a disposizione, più sono in grado di adattarsi alle diverse esigenze in modo autonomo. La conseguenza di tutto ciò è che è possibile realizzare modelli per l'analisi di dati più grandi e complessi, elaborando risultati velocemente e applicabili su larga scala.

Guardando il Machine Learning da una prospettiva puramente informatica, anziché scrivere il codice di programmazione attraverso il quale, passo dopo passo, si dice alla macchina cosa fare, al computer vengono forniti solo dei set di dati inseriti in un generico algoritmo che sviluppa una propria logica per svolgere la funzione, l'attività ed il compito richiesti. [5]

A seconda del tipo di algoritmo utilizzato per permettere l'apprendimento alla macchina, cioè in base alle modalità con cui la macchina è in grado di imparare e accumulare dati, si possono suddividere tre differenti sistemi di apprendimento automatico: [7]

- Supervisionato: esso consiste nel fornire al sistema informatico della macchina una serie di nozioni specifiche e codificate, ossia di modelli ed esempi che permettono di costruire un vero e proprio database di informazioni e di esperienze. In questo modo, quando la macchina si trova di fronte ad un problema, non dovrà fare altro che attingere alle esperienze inserite nel proprio sistema, analizzarle, e decidere quale risposta dare sulla base di esperienze già codificate. Questo tipo di apprendimento è, in qualche modo, fornito già confezionato e la macchina deve essere solo in grado di scegliere quale sia la migliore risposta allo stimolo che le viene dato. Il campo di applicabilità dell'apprendimento supervisionato è molto vasto e va da quello medico a quello di identificazione vocale.
- Non supervisionato o senza supervisione: esso invece prevede che le informazioni inserite all'interno della macchina non siano codificate, ossia la macchina ha la possibilità di attingere a determinate informazioni senza avere alcun esempio del loro utilizzo e, quindi, senza avere conoscenza dei risultati attesi a seconda della scelta effettuata. Dovrà essere la macchina stessa, quindi, a catalogare tutte le informazioni in proprio possesso, organizzarle ed imparare il loro significato, il loro utilizzo e, soprattutto, il risultato a cui esse portano. L'apprendimento senza supervisione offre maggiore libertà di scelta alla macchina che dovrà organizzare le informazioni in maniera intelligente e imparare quali siano i risultati migliori per le differenti situazioni che si presentano.

- Per rinforzo: rappresenta probabilmente il sistema di apprendimento più complesso. Esso prevede che la macchina sia dotata di sistemi e strumenti in grado di migliorare il proprio apprendimento e, soprattutto, permette di comprendere le caratteristiche dell'ambiente circostante. In questo caso, quindi, alla macchina vengono forniti una serie di elementi di supporto, ad esempio sensori e telecamere al fine di rilevare quanto avviene nell'ambiente circostante ed effettuare scelte per un migliore adattamento all'ambiente che la circonda. Questo tipo di apprendimento è tipico delle auto senza pilota.

Per garantire la massima performance e il miglior risultato possibile per la risposta agli stimoli esterni, a seconda ovviamente della macchina su cui si deve operare, i tre modelli di apprendimento vengono utilizzati in maniera differente a seconda dei casi.

1.4 Reti neurali

Le reti neurali sono indispensabili per risolvere problemi ingegneristici di Intelligenza Artificiale, Machine Learning e Deep Learning. Esse sono modelli matematici composti da neuroni artificiali che si ispirano al funzionamento biologico del cervello umano e danno vita a modelli basati sulle interconnessioni delle informazioni in quanto sono formati da un corpo centrale e da terminazioni di ingresso e di uscita. I neuroni, anche chiamati nodi o unità della rete neurale, vengono solitamente dislocati in livelli, che come possiamo vedere nella *figura 4* sono di tre tipi: [8]

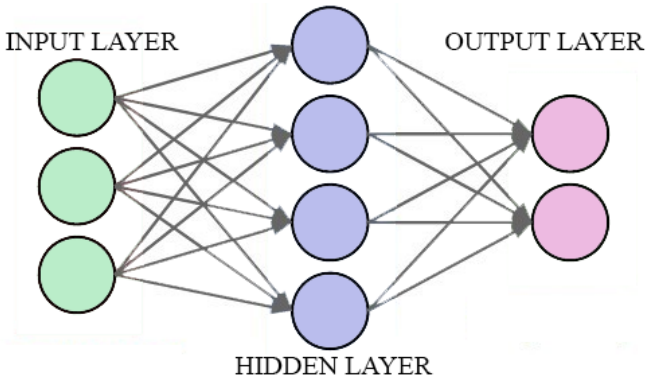

Figura 2 - I livelli di interconnessione dei nodi

- Livello di ingresso o input layer: è il livello progettato per ricevere le informazioni provenienti dall'esterno al fine di imparare, riconoscere e processare le stesse informazioni ricevute.
- Livello nascosto o hidden layer: collega il livello di ingresso con quello di uscita e aiuta la rete neurale ad imparare le relazioni complesse analizzate dai dati.
- Livello di uscita o output layer: è il livello finale che mostra il risultato di quanto il programma è riuscito a imparare.

La struttura dei neuroni è dotata di parallelismo, ovvero le parti che la costituiscono non lavorano in cascata ma in modo indipendente l'una dall'altra. Le informazioni non sono distribuite in un'unica memoria centrale ma in tutti i vari nodi della rete: in questo modo la rete è estremamente robusta. Questo procedimento è l'esatto opposto di quello che viene applicato ad esempio, nell'informatica tradizionale, dove i dati vengono memorizzati in un'unica memoria centrale e i calcoli vengono effettuati in modo seriale e non in parallelo. L'obiettivo dello studio delle reti neurali consiste nel trovare i valori ottimali dei pesi presenti sulle connessioni in modo da raggiungere lo scopo che ci si è prefissati. [8]

La storia dei modelli neurali nasce nel 1943 quando McCulloch e Pitts propongono il primo modello di neurone. Secondo questi studiosi infatti, il nucleo pensante del neurone è rappresentato da una porta logica che può assumere due soli stati interni, "acceso" o "spento". Inoltre, ciascun neurone è dotato di un insieme di ingressi attraverso i quali riceve gli stimoli dall'ambiente circostante e/o i responsi prodotti dai neuroni adiacenti. La somma pesata di tali impulsi, confrontata con un'opportuna soglia interna al neurone, ne determina lo stato finale ed il conseguente segnale in uscita. Intorno al 1950, però Von Neumann fece un'obiezione che determinò da quel momento in poi un vero e proprio cambio di rotta nello studio di questo campo: infatti, se i neuroni vengono considerati come singoli elementi, mancano di robustezza.

Così, nel 1954 Cragg e Temperly svilupparono il concetto di "*insieme o pool di neuroni*" in cui veniva data importanza al comportamento non più del singolo neurone, ma a quello collettivo delle cellule aggregate. Nel 1960, grazie a Frank Rosenblatt, nasce il percettrone, una rete con uno strato di ingresso ed uno di uscita ed una regola di apprendimento intermedia basata sulla minimizzazione degli errori. Gli anni compresi tra il 1969 e il 1985 furono i cosiddetti anni della crisi, in quanto si evidenziarono i limiti fondamentali delle architetture a cella singola derivandone chiaramente la ristrettezza del campo applicativo. Tuttavia, sempre negli stessi anni, fu approfondito lo studio delle memorie autoassociative. Nel 1986, David Rumelhart introdusse il terzo strato delle reti neurali, quello oggi chiamato strato H – hidden. Si iniziò quindi a parlare di un percettrone multistrato, in quanto vennero identificati i modelli di apprendimento per addestrare le reti MLP – Multi-Layers Perceptron. Rumelhart, inoltre, propose alla comunità scientifica il cosiddetto algoritmo di retropropagazione dell'errore anche detto error backpropagation

che vedremo nel capitolo successivo. Da quel momento in poi si sono fatti grandi passi in avanti, ad esempio anche grazie alla nascita delle reti ricorrenti, che simulano il comportamento dei sistemi dinamici. [9]

Le reti neurali si possono suddividere in due grandi categorie: [10]

- Supervisionate: di questa categoria fanno parte i problemi di classificazione, riconoscimento, diagnosi e regressione. Per tali problemi vengono utilizzate reti neurali come le reti percettroni e le reti adaline.
- Non supervisionate: di quest'altra categoria invece, fanno parte i problemi di clustering e di indirizzamento della memoria e le reti usate per risolvere questo tipo di problemi, risultano essere quelle Hopfield e self organizing maps.

Per quanto riguarda l'architettura di una rete neurale, essa è formata, come già detto, da un set di neuroni collegati fra di loro e ne esistono tre differenti tipi: [10]

- Single-layer feed-forward: in questo caso i dati si propagano in un unico senso entrando in un punto ed uscendo da un altro e non vi è mai la presenza di cicli.
- Multi-layer feed-forward: in questo secondo caso il funzionamento è analogo al primo ma invece di esserci un singolo livello, vi sono più livelli hidden ovvero nascosti.
- Ricorrenti: in quest'ultimo caso una parte delle informazioni torna indietro al neurone di input e sono dunque caratterizzate dalla presenza di cicli e da dinamiche temporali.

Nel paragrafo dell'Intelligenza Artificiale abbiamo parlato dei sistemi esperti, ovvero delle applicazioni in grado di riprodurre artificialmente le prestazioni di un professionista di un determinato dominio di conoscenza. Erroneamente essi vengono spesso associati alle reti neurali artificiali, in quanto il principio su cui si basano questi due concetti sembra essere abbastanza simile, ma in realtà vi sono delle differenze evidenti. Ad esempio, un sistema esperto può dedurre dei ragionamenti, mentre la rete neurale non è in grado di fare ciò ma riesce ad esempio, a riconoscere un volto umano in un'immagine molto complessa. Inoltre, un sistema esperto necessita di un intervento iniziale da parte di un professionista del campo trattato che definisca le regole iniziali che alimentano il motore inferenziale, mentre le reti neurali deducono regole e attività in modo automatico tramite l'apprendimento. In sintesi, quindi, le reti neurali rientrano a pieno titolo fra le tecniche di Intelligenza Artificiale, in quanto costituiscono il naturale complemento dei sistemi esperti

con cui dovranno integrarsi nei futuri sistemi intelligenti. Esse infatti all'interno dell'A.I. si collocano nel settore dell'Apprendimento Automatico, ovvero nel già trattato Machine Learning. [11]

1.5 Supervised Learning

L'apprendimento supervisionato è una tecnica di apprendimento automatico che mira ad istruire un sistema informatico in grado di elaborare automaticamente previsioni sui valori di uscita di un sistema rispetto ad un input basandosi su una serie di esempi ideali che gli vengono inizialmente forniti, costituiti da coppie di input e di output.

In particolare, esso si basa sull'addestramento di data sample proveniente da una data source che presenta già una classificazione corretta assegnata. Tali tecniche sono utilizzate nei modelli feedforward o MultiLayer Perceptron (MLP). Gli MLP hanno tre caratteristiche distintive:

- Uno o più strati di neuroni nascosti che non fanno parte degli strati di input o output della rete che consentono alla rete di apprendere e risolvere problemi complessi
- La non linearità riflessa nell'attività neuronale è differenziabile
- Il modello di interconnessione della rete mostra un alto grado di connettività

Queste caratteristiche insieme all'apprendimento attraverso la formazione risolvono problemi difficili e diversificati.

L’apprendimento che avviene attraverso l'addestramento in un modello ANN supervisionato, viene chiamato anche “Error Back-propagation Algorithm”. L'algoritmo di correzione dell'errore addestra la rete in base ai campioni input-output e trova il segnale di errore, che è rappresentato dalla differenza tra l'output calcolato e l'output desiderato. Inoltre, regola i pesi sinaptici dei neuroni che sono proporzionali al prodotto del segnale dell’errore e all'istanza di input del peso sinaptico.

Sulla base di questo principio, l'apprendimento della propagazione all'indietro dell'errore avviene in due passaggi: [32]

- Forward Pass: qui, il vettore di input viene presentato alla rete. Questo segnale di ingresso si propaga in avanti, neurone per neurone attraverso la rete ed emerge all'estremità di uscita della rete come segnale di uscita: $y(n) = \varphi(v(n))$ dove $v(n)$ è il campo locale indotto di un neurone definito da $v(n) = \Sigma\ w(n)y(n)$. L'output calcolato sullo strato di output $o(n)$ viene confrontato con la risposta desiderata $d(n)$ e trova l'errore $e(n)$ per quel neurone. I pesi sinaptici della rete durante questo passaggio rimangono gli stessi.
- Backward Pass: il segnale di errore originato dal neurone di uscita del livello viene propagato all'indietro attraverso la rete. Questo calcola il gradiente locale per ogni neurone in ogni strato e consente ai pesi sinaptici della rete di subire modifiche secondo la regola delta: $w(n) = \eta * \delta(n) * y(n)$.

Questo calcolo ricorsivo continua, con il passaggio in avanti seguito dal passaggio all'indietro per ogni modello di input fino a quando la rete non è convergente.

Il paradigma di apprendimento supervisionato di una ANN è efficiente e trova soluzioni a diversi problemi lineari e non lineari come la classificazione, il plant control, il forecasting, e la robotica.

1.6 Unsupervised Learning

L'apprendimento non supervisionato utilizza algoritmi di apprendimento automatico per analizzare e raggruppare set di dati non etichettati. Questi algoritmi scoprono modelli nascosti o raggruppamenti di dati senza la necessità dell'intervento umano. La capacità di scoprire somiglianze e differenze nelle informazioni, lo rendono ideale per l'analisi esplorativa dei dati, per le strategie di vendita incrociata, segmentazione dei clienti e per il riconoscimento delle immagini.

La mancanza di direzione per questo tipo di algoritmo può a volte essere vantaggiosa, poiché consente all'algoritmo di cercare modelli che non sono stati precedentemente considerati.

Le principali caratteristiche delle Self-Organizing Maps (SOM) sono: [32]

- Trasforma un modello di segnale in entrata di dimensione arbitraria in una mappa dimensionale o bidimensionale ed esegue questa trasformazione in modo adattivo.

- La rete rappresenta una struttura feedforward con un singolo strato computazionale costituito da neuroni disposti su righe e colonne.
- In ogni fase della rappresentazione, ogni segnale in ingresso è mantenuto nel suo contesto appropriato.
- I neuroni che si occupano di informazioni strettamente correlate sono vicini tra loro e comunicano attraverso connessioni sinaptiche.

Lo strato computazionale è anche chiamato strato competitivo poiché i neuroni nello strato competono tra loro per diventare attivi. Quindi, questo algoritmo di apprendimento è chiamato “competitive algorithm”.

L'algoritmo non supervisionato in SOM funziona in tre fasi:

- Competition phase: per ogni pattern di input x presentato alla rete, viene calcolato il prodotto interno con il peso sinaptico w. I neuroni nello strato competitivo trovano una funzione discriminante che induce competizione tra i neuroni e il vettore di peso sinaptico che risulta più vicino al vettore di input (basandosi sul calcolo della distanza euclidea) è annunciato come vincitore della competizione. Quel neurone è chiamato “best matching neuron”, cioè $x=\text{argmin}\|x-w\|$.
- Cooperative phase: il neurone vincente determina il centro di un intorno topologico h di neuroni cooperanti. Ciò viene eseguito dall'interazione laterale d tra i neuroni cooperativi. Questo “topological neighborhood” riduce le sue dimensioni nel tempo.
- Adaptive phase: in questa fase al neurone vincente e ai neuroni vicini è consentito di aumentare i propri valori individuali della funzione discriminante in relazione al pattern di input e attraverso opportuni aggiustamenti del peso sinaptico, $\Delta w = \eta h(x)(x - w)$.

Dopo ripetute presentazioni dei pattern di addestramento, i vettori di peso sinaptico tendono a seguire la distribuzione dei pattern di input a causa dell'aggiornamento dei neuroni vicini e quindi la ANN apprende senza supervisore. [32]

1.7 Deep Learning

Il Deep Learning, la cui traduzione letterale significa apprendimento profondo, è una sottocategoria del Machine Learning e indica quella branca dell'Intelligenza Artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate Reti Neurali Artificiali. Esso, attraverso sistemi artificiali, permette di simulare i processi di apprendimento del cervello biologico per insegnare alle macchine non solo ad apprendere autonomamente, ma a farlo in modo più "profondo", proprio come sa fare il cervello umano.

Si è iniziato a parlare di un approccio profondo solamente nel 2006, quando Geoffrey Hinton, pioniere del campo delle ANN, ed i suoi colleghi si sono imbattuti a descrivere la creazione di una rete neurale con molti più livelli di quelli a cui si era solitamente abituati. Successivamente, nel 2009 fu sviluppata una prima reale applicazione di successo, ovvero il riconoscimento vocale, di questo nuovo trend che fu denominato "Deep Learning". Il 2012 è una data fondamentale, in quanto in quell'anno vengono presentati gli straordinari risultati ottenuti in un esperimento anche noto come "*l'esperimento del gatto*".

L'esperimento era volto al riconoscimento visivo, effettuato tramite software prova, in un settore fino ad allora arduo e mai esplorato: la capacità di distinguere 1000 categorie visive. Nel corso di questo esperimento, il pool di studiosi presentò questa nuova tecnologia che aveva consentito alla macchina di riconoscere uomini e animali tramite il confronto con milioni di altre immagini, senza il bisogno di un intervento codificato da parte dell'uomo. Al contempo però, i risultati del test furono di difficile interpretazione per il team di Mountain View: alcuni neuroni, situati nel livello più alto della rete, mostrarono un'energica risposta alle immagini in cui erano presenti gatti (da qui il curioso nome), altri risposero alle immagini di volti umani. Non venne invece trovato alcun neurone che reagisse alle immagini di auto, presenti in larga misura nel database adoperato. Nel 2015 invece, durante ImageNet vennero presentati dei risultati clamorosi. Ciò che lasciò tutti di stucco fu la realizzazione di un Deep Learning su 152 livelli di astrazione contro i 30 su cui ci si era basati solitamente prima di questa ricerca. Fu un risultato notevole anche considerato il fatto che più sono i livelli di astrazione, maggiore è la capacità di apprendimento, ovvero la macchina è più "intelligente". [12] [13]

Le reti neurali artificiali usate dal Deep Learning sono le Deep Neural Network (DNN), ovvero le reti neurali profonde composte da più livelli organizzati gerarchicamente. Le reti neurali artificiali tradizionali, ovvero le Artificial Neural Networks (ANN), normalmente contengono due o tre layers, mentre le DNN, come si può vedere nella *figura 5*, hanno un numero di livelli tendenzialmente compreso tra 7 e 50. È possibile infatti che ne contengano anche di più, addirittura più di 100: aumentando il numero di livelli, le prestazioni della rete risultano essere migliori a discapito però dell'efficienza. [14]

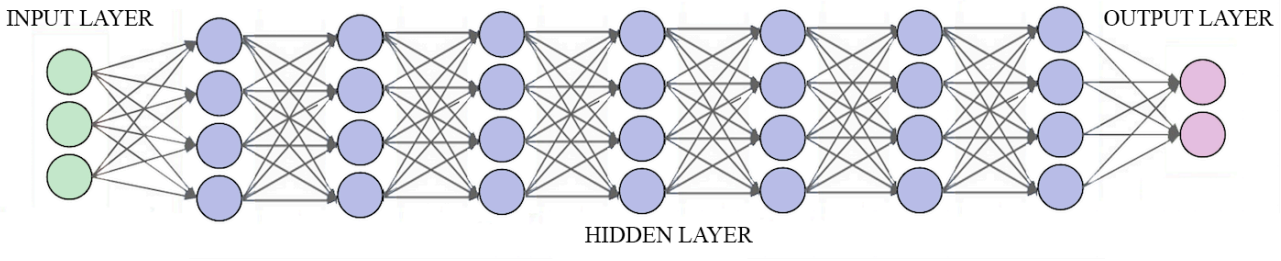

Figura 3 - I livelli di interconnessione dei nodi di una DNN

Le reti neurali profonde su cui si basa il Deep Learning, sono reti che risultano avere una complessità computazionale davvero notevole. Essa è influenzata non soltanto dalla profondità, ovvero dal numero di livelli, ma anche da: [10]

- Numero di neuroni e di connessioni stabilite: un elevato numero di neuroni e di conseguenza delle relative connessioni tra questi ultimi, renderà il cammino sulla rete più costoso poiché aumenterà il numero di operazioni necessarie per completarlo.
- Pesi della rete: più aumenta il numero di pesi, ovvero più crescono i parametri che si vogliono apprendere, più elevata sarà ovviamente la complessità della DNN.

Se dal punto di vista delle potenzialità il Deep Learning può sembrare più affascinante e utile del Machine Learning, va precisato che il calcolo computazionale richiesto per il suo funzionamento è davvero impattante, anche dal punto di vista economico. Infatti, fino a poco tempo fa, le reti neurali artificiali sono state ignorate sia dalla comunità della ricerca che dall'industria a causa del loro costo computazionale. Per reggere il carico di lavoro di un sistema di Deep Learning, sono necessarie CPU molto avanzate e GPU top di gamma che costano ancora migliaia di dollari.

Una soluzione a questo problema potrebbe essere il ricorso a capacità computazionali via Cloud, ma esse lo attenuano solo in parte perché la formazione di una rete neurale profonda

richiede molto spesso l'elaborazione di grandi quantità di dati che necessitano dunque di GPU di fascia alta per molte ore. Tuttavia, tra il 2006 e il 2012, il gruppo di ricerca guidato da Geoffrey Hinton dell'Università di Toronto è stato in grado finalmente di parallelizzare gli algoritmi per le ANN su architetture parallele. Il principale risultato è stato un notevole incremento del numero di strati, neuroni e parametri del modello in generale (anche oltre i 10 milioni di parametri) permettendo alle macchine di computare una quantità massiccia di dati addestrandosi su di essi. Pertanto, risulta fondamentale avere a disposizione training set molto grandi per elaborare un modello di Deep Learning, requisito che risulta essere molto adatto per affrontare l'era dei Big Data. [13] [15]

Seppur come si è appena potuto constatare la richiesta di capacità computazionali enormi da parte delle DNN possa rappresentare un limite, c'è un fattore che fa preferire un meccanismo di Deep Learning a discapito di uno di Machine Learning: la scalabilità. Essa va quindi intesa come la capacità di adattarsi ad un aumento di domanda o di carico di lavoro, cioè indica se un sistema ha possibilità di crescita o meno. I sistemi di Deep Learning, infatti, continuano a migliorare le proprie prestazioni all'aumentare dei dati forniti, mentre le applicazioni di Machine Learning, una volta raggiunto un certo livello di performance, non sono più scalabili nemmeno aggiungendo esempi e dati di training alla rete neurale. Infatti, nei sistemi di Machine Learning, le caratteristiche di un determinato oggetto vengono estratte e selezionate manualmente e servono per creare un modello in grado di categorizzare gli oggetti [13]. Nei sistemi di Deep Learning invece, l'estrazione delle caratteristiche avviene in modo automatico: la rete neurale apprende in modo autonomo come analizzare dati grezzi e come svolgere un compito. Nella *figura 6* si può comunque osservare uno schema riassuntivo in cui si evidenzia quando è preferibile utilizzare un algoritmo di Machine Learning e quando invece è meglio applicarne uno di Deep Learning:

	Machine Learning	Deep Learning
Dimensioni del dataset	Prestazioni eccellenti su un dataset di piccole o medie dimensioni	Prestazioni eccellenti su un dataset di grandi dimensioni
Potenza dell'hardware	Necessità di una macchina di fascia bassa	Necessita di una macchina molto potente

Tempo di esecuzione	Da pochi minuti a diverse ore	Fino a diverse settimane
Interpretabilità	Alcuni algoritmi sono relativamente facili da interpretare	Algoritmi di difficile interpretazione

Figura 4 - ML vs DL

La fase di implementazione vera e propria del Deep Learning necessita di diversi step:

- Elaborazione dei dati
- Creazione delle features
- Creazione della rete neurale
- Evoluzione e allenamento della rete creata

Una volta lanciato il Deep Learning, dovrà essere messa in atto una fase di analisi e aggiustamento. Questa fase prevede:

- Aggiustamento delle features scelte
- Inclusione di dati aggiuntivi
- Analisi dei risultati predittivi
- Implementazione delle predizioni

Il Machine Learning Specialist dunque, aggiusta le features scelte per capire se sono effettivamente quelle che possono essere utili all'obiettivo da raggiungere e vengono valutati i dati disponibili ed eventualmente se ne aggiungono di nuovi. Si analizzano poi i risultati ottenuti dall'analisi e, se i dati sono soddisfacenti e i risultati sono quelli che ci si aspettava, si implementano le predizioni all'interno delle strategie di business aziendali. [16]

Ci sono diversi modelli di reti profonde che si possono suddividere in: [10]

- Modelli con training supervisionato: ad esempio CNN, ovvero le Convolutional Neural Network.
- Modelli con training non supervisionato: ad esempio RBM, cioè la Restricted Boltzmann Machine.
- Modelli ricorrenti: ad esempio RNN (Recurrent Neural Networks) e LSTM (Long Short-Term Memory) che verranno trattati nello specifico nel prossimo capitolo.

Capitolo 2: Recurrent Neural Networks (RNN)

2.1 Introduzione alle RNN

Come si è già visto nel capitolo precedente, ci sono diversi modelli di reti profonde. In questo capitolo tratteremo le reti neurali ricorrenti.

Una rete neurale ricorrente (recurrent neural network, RNN) è una classe di rete neurale artificiale in cui i valori di uscita di uno strato di un livello superiore vengono utilizzati come ingresso ad uno strato di livello inferiore. Quest'interconnessione tra strati permette l'utilizzo di uno degli strati come memoria di stato, e consente, fornendo in ingresso una sequenza temporale di valori, di modellarne un comportamento dinamico temporale dipendente dalle informazioni ricevute agli istanti di tempo precedenti. Ciò le rende applicabili a compiti di analisi predittiva su sequenze di dati, quali possono essere ad esempio il riconoscimento della grafia o il riconoscimento vocale. [17]

Come si può dedurre dal significato della parola stessa “ricorrente”, ovvero un qualcosa che si verifica spesso o ripetutamente, questo tipo di rete neurale è chiamata in questo modo in quanto esegue la medesima operazione su insiemi di input sequenziali. L'idea, infatti, alla base di RNN è quella di utilizzare le informazioni sequenziali. In una rete neurale tradizionale, assumiamo che tutti gli input siano indipendenti l'uno dall'altro, ma per molte applicazioni nella realtà questo è tutt’altro che un vantaggio. Ad esempio, se si vuole fare una previsione in una frase su quale sarà la prossima parola che verrà scritta, è fondamentale sapere quali parole sono venute prima. Le RNN, a differenza delle reti neurali tradizionali, eseguono lo stesso compito per ogni elemento di una sequenza, con l'output che dipende fortemente dai calcoli precedenti. In altre parole, si può pensare che le RNN abbiano una sorta di “memoria” in cui viene immagazzinato tutto ciò che è stato calcolato fino a quel momento. Anche se teoricamente le RNN possono fare uso di informazioni in sequenze notevolmente lunghe, le informazioni contenute in questa “memoria”, non vengono conservate per sempre e non è dunque sempre possibile ad ogni passo accedere e verificare tutti gli step precedenti. [18]

Il training di una RNN è simile a quello di una rete neurale tradizionale. Anche qui viene utilizzato l'algoritmo di Back-Propagation, ma con una modifica. In questo algoritmo, si

confronta il valore in uscita del sistema con il valore desiderato e sulla base della differenza così calcolata ovvero l'errore, l'algoritmo modifica i pesi sinaptici della rete neurale. [19] La retroazione dunque, dovrebbe consentire al sistema di ridurre progressivamente l'errore cioè di effettuare la cosiddetta minimizzazione del gradiente, in modo tale da far convergere progressivamente il set dei valori di uscita verso quelli desiderati. Poiché nelle RNN i parametri sono condivisi da tutte le fasi temporali della rete, il gradiente di ciascuna uscita dipende non soltanto dai calcoli del passo temporale corrente, ma anche dai passi temporali precedenti. Questo è chiamato Backpropagation Through Time (BPTT). Come già detto le RNN gestite con BPTT hanno difficoltà ad imparare sequenze a lungo termine a causa del cosiddetto problema del gradiente che svanisce o esplode (vanishing/exploding gradient). Esistono alcuni meccanismi per affrontare questi problemi e alcuni tipi di RNN (come LSTM tratteremo nella parte seguente di questo capitolo) sono stati progettati specificamente per aggirarli. [18]

2.2 CNN vs RNN

Fino ad ora abbiamo approfondito lo studio delle RNN, ma esiste un altro tipo di reti profonde, ovvero le CNN che sono molto diffuse. Le Convolutional Neural Networks (CNN) sono un tipo di reti neurali utilizzate, ad esempio, nel campo della Computer Vision. Il nome deriva dal tipo dei livelli nascosti in cui consistono, ovvero: [20] [21]

- Convoluzione: è ispirato dai processi biologici per l'analisi visiva negli organismi viventi. Lo strato di neuroni che si occupa della convoluzione divide l'immagine in vari frammenti sovrapposti, che sono in seguito analizzati per individuare le particolarità che lo caratterizzano. L'informazione viene trasferita allo strato seguente sotto forma di una feature map e contiene le relazioni tra neuroni e particolarità.
- Pooling (subsampling): è un processo alquanto comune nelle reti neurali che consiste nel ridurre la dimensione dei dati in modo da rendere più rapida l'analisi senza perdere troppa precisione. Ad esempio, si può pensare al procedimento che viene attuato per ridurre la qualità di un'immagine, ovvero alla diminuzione della quantità di pixel da cui è composta. Una regione di pixel diventa un unico pixel al quale è assegnato un colore in base alla media dei colori della sua regione.
- Normalizzazione: è anch'essa largamente utilizzata per evitare le anomalie in seguito a vari passaggi negli strati della rete neurale. Le funzioni più utilizzate sono

la Rectified Linear Units (ReLU), la tangente iperbolica e la funzione sigma. Tra queste si preferisce usare ReLU in quanto è considerata la più rapida.

- Strato di connessione totale (full connection layer): è l'ultimo strato nascosto della rete neurale, nel quale tutti gli input dei vari neuroni sono messi insieme permettendo quindi alle particolarità di essere identificate.
- Strato di apprendimento (loss layer): è lo strato finale per tutte le reti neurali e permette al sistema di modificare i valori associati ai neuroni sulla base della correttezza dei risultati emessi. Il metodo di allenamento più spesso utilizzato è la Back-Propagation.

Alla luce di quello che è stato detto fino ad ora, possiamo dunque individuare delle differenze tra le CNN e le RNN. Innanzitutto, la CNN è una rete neurale feed-forward che viene generalmente utilizzata per il riconoscimento dell'immagine e la classificazione degli oggetti. RNN invece, lavora sul principio del salvataggio dell'output di un layer e lo reinserisce nell'input per prevederne l'output. CNN inoltre, considera solo l'ingresso corrente mentre RNN considera sia l'ingresso corrente che gli input ricevuti in precedenza, memorizzandoli dunque grazie alla sua memoria interna. Alla luce di ciò, possiamo dunque concludere che anche il campo di applicazione delle due reti risulta essere diverso. Gli RNN vanno quindi bene per trattare una serie di dati e vengono utilizzati molto nei problemi che devono risolvere il quesito "che cosa succederà dopo?" mentre le CNN sono particolarmente adatte alla risoluzione di problemi come la classificazione delle immagini. [20] [22]

2.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) è una particolare architettura di rete neurale ricorrente, ovvero di RNN, in grado di trattare le dipendenze a lungo termine. È stata introdotta da Hochreiter e Schmidhuber nel 1997 ma il suo potenziale è stato scoperto soltanto negli ultimi anni in quanto questa rete risulta essere particolarmente adatta per risolvere gli algoritmi di Deep Learning. Essa infatti, permette di ovviare ad alcuni problemi riscontrati nelle RNN tradizionale come ad esempio il "vanishing/exploding gradient".

Il gradiente che svanisce o che esplode, infatti, è un grossissimo limite delle RNN. Per provare a spiegarlo, consideriamo un modello linguistico che tenta di prevedere la parola successiva in base a quelle precedenti. Prendiamo in considerazioni due frasi "le nuvole sono nel *cielo*" e "Sono nato in Francia … Io parlo in modo fluente *francese*". Nel primo

caso il divario, cioè il gradiente, presente tra la parola da prevedere “cielo” e le parole su cui devo basarmi per effettuare la previsione ovvero, ad esempio “nuvole”, è molto piccolo, per cui una RNN è perfettamente in grado di svolgere il suo lavoro. Nel secondo caso invece, per prevedere “francese” posso utilizzare le parole più vicine come, ad esempio, “parlo” che mi suggerisce sicuramente il nome di una lingua, ma per contestualizzare quale tra le molte che esistono nel mondo ho bisogno assolutamente della parola “Francia” che si trova ben più lontano da “francese” rispetto a dove si trova “parlo”. Il divario quindi tra le parole che si stanno considerando è molto più ampio e le RNN all’aumentare del gradiente diventano sempre meno capaci a connettere queste informazioni. [23]

Tuttavia, non solo il vanishing/exploding gradient rappresenta un limite per una RNN. Un altro caso in cui una RNN risulta performare non in modo efficace è quando nella sequenza sono presenti informazioni inutili e dunque da scartare. Ad esempio, durante l'analisi di una pagina web, potrebbe esserci un codice HTML ausiliario che è irrilevante ai fini della valutazione della sequenza trasmessa sulla pagina. Sarebbe utile quindi, avere un meccanismo che sia predisposto per saltare tali simboli nell’analisi che si sta svolgendo.

Se consideriamo, invece, una situazione in cui vi è un’interruzione logica tra le parti di una sequenza come, ad esempio, una transizione tra i capitoli in un libro che risultano essere logicamente molto diversi tra di loro, sarebbe molto utile un metodo che reimpostasse la nostra analisi sulla base della nuova logica senza intervenire sugli step passati e dunque senza doverla riiniziare da capo [24].

Per risolvere questi problemi sono stati proposti diversi metodi tra cui la Long Short Term Memory (LSTM).

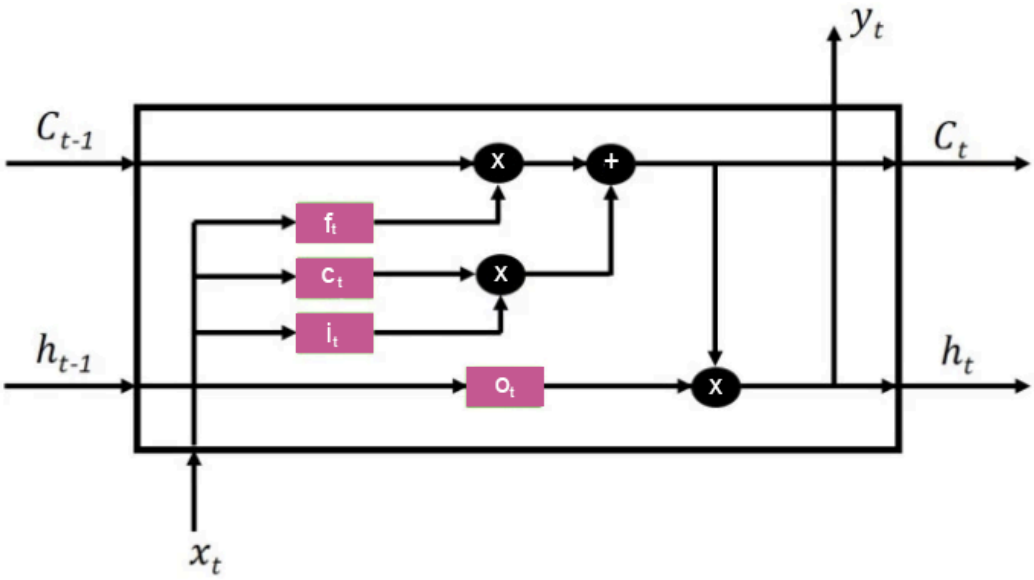

Figura 5 - Architettura di una LSTM

Nella *figura 7* viene mostrata l'architettura di una LSTM. Una rete LSTM è composta da celle (LSTM blocks) concatenate fra loro. Ogni cella è a sua volta composta da quattro tipi di porte: [25]

- I(t) – input gate: controlla quanto nuovo contenuto dovrebbe essere aggiunto alla cella di memoria C(t). Prende in considerazione sia l'output dello stato nascosto precedente, ovvero h(t-1) sia il valore di input corrente x(t). Poi, attraverso una funzione sigma, emette un valore compreso tra 0 e 1, e decide se aprire o chiudere la connessione con la valvola "X" per tramandare o meno l'informazione.
- C(t) – candidate gate: i valori C(t) sono i candidati per essere messi in memoria. In questo gate vengono prese entrambe le informazioni sullo stato precedente h(t-1) e vengono combinate con il valore di input corrente x(t) usando una funzione di attivazione "tanh" che assicurerà che i valori oscillino tra -1 e 1. Dopo l'attivazione, i valori candidati si dirigeranno verso la valvola "X" dove avverrà una "moltiplicazione". Se la porta di ingresso i(t) emette un valore pari a 1, la valvola consente sostanzialmente il passaggio di tutti i valori di uscita candidati. Se, invece, il valore è 0,5, allora solo metà delle informazioni potrà effettuare il passaggio. Successivamente i valori passeranno attraverso la valvola di addizione "+" per unirsi agli altri valori in memoria.
- F(t) – forget gate: in questa porta ci si pone il quesito chiave di tutto il meccanismo delle LSTM. "*Tutti i dati che fino ad ora sono stati messi in memoria sono*

importanti o possiamo dimenticarne qualcuno?" Per rispondere a ciò, è necessario un forget gate che è in grado di dirmi se il dato in memoria C(t-1) deve essere dimenticato o meno. Ancora una volta vengono prese in considerazione entrambe le informazioni sullo stato precedente h(t-1) e il valore di input corrente x(t). L'attivazione è nuovamente gestita da una funzione sigma, che controllerà che i valori siano compresi tra 0 e 1: se il valore risulterà essere 1, allora sarà permesso a tutti i dati precedenti in memoria C(t-1) di passare.

- O(t) – output gate: questa porta controlla quanto contenuto della memoria bisogna cedere allo stato nascosto, ovvero alla cella successiva. Nuovamente contiene sia l'informazione di stato precedente h(t-1) che il valore di input corrente x(t) e l'attivazione, come nel caso precedente, è gestita da una funzione sigma.

Come si è appena potuto constatare, quasi la totalità delle porte analizzate fa uso di una funzione di attivazione sigmoidale che può assumere valori compresi tra 0 e 1, dove 0 indica la totale inibizione mentre 1 la totale attivazione. In sintesi, se i(t) non viene attivato, la cella mantiene lo stato precedente, mentre se è abilitato, lo stato corrente verrà combinato con il valore in ingresso. F(t), invece, resetta lo stato corrente della cella e o(t) decide se il valore all'interno della cella dev'essere portato verso l'uscita o meno.

Alla luce di ciò, si può dunque affermare che le celle LSTM sono in grado di ricordare le informazioni per un tempo indefinito.

Capitolo 3: Sequence Analysis

3.1 Big Data e Business Intelligence

I Big Data sono definiti come una raccolta di dati informativi molto estesa in termini di: [30]

- Volume: il volume dei dati aumenta in modo esponenziale al trascorrere del tempo.
- Velocità: i dati vengono generati molto velocemente.
- Varietà: i dati arrivano in diversi formati, tipi e strutture.
- Veridicità: i dati devono essere di qualità.
- Valore: per ottenere informazioni utili, è necessario tradurre i dati in un'applicazione di business.

In altre parole, il termine si riferisce ad un insieme di dataset estremamente grandi e talmente complessi da rendere impossibile la gestione degli stessi con strumenti tradizionali. Essi vengono utilizzati per diversi scopi, come ad esempio quello di misurare le prestazioni di un'organizzazione nonché di un processo aziendale.

Alla luce di ciò, è possibile individuare delle differenze con la Business Intelligence sia in materia di dati che nel loro utilizzo:

- La Business Intelligence utilizza la statistica descrittiva con dataset limitati, dati puliti e modelli semplici. [31]
- Big Data utilizza la statistica inferenziale e concetti di identificazione di sistemi non lineari per dedurre leggi da dataset eterogenei, dati grezzi e modelli predittivi e complessi. [31]

3.2 Data Mining

Data Mining è quel processo che, a partire da grandi quantità di dati memorizzati su database, datawarehouse o altri repository, è in grado di estrarre contenuto informativo interessante. Per interessante va inteso tutto ciò che prima non era conosciuto e che non è già esplicitamente contenuto nei dati utilizzati.

Data mining è inoltre definito come un sottoprocesso del Knowledge Discovery in Databases (KDD). I passaggi che vengono generalmente fatti nel KDD sono:

- Definizione dell'obiettivo finale e del dominio applicativo
- Estrazione della totalità dei dati grezzi o di un sottoinsieme di essi
- Preprocessing: in questa fase vengono eliminati i dati rumorosi e vengono invece formattati i rimanenti in modo da avere uniformità nell'analisi.
- Definizione delle features (o attributi) e dei records: in questo step vengono ridotti i dati ad esempio tramite tecniche di features selection/extraction per mantenere solo i dati ritenuti maggiormente interessanti per l'analisi.
- Task di Data Mining: alla luce degli obiettivi definiti precedentemente, possono essere applicati diversi task come Clustering e Regole di associazione che produrranno una serie di pattern, ovvero di informazione utile per gli obiettivi definiti precedentemente.
- Interpretazione e valutazione: l'informazione estratta può essere utilizzata ad esempio per implementare una possibile applicazione di business.

In particolare, gli algoritmi di Data Mining possono essere descrittivi o preditivi. Quelli descrittivi identificano pattern e relazioni tra i dati. Tra essi troviamo:

- Clustering: in esso i dati considerati dall'algoritmo simili vengono raggruppati
- Summarizations: ai dati mappati in sottoinsiemi viene associata una descrizione
- Regole di associazione: vengono ricercate delle relazioni di causa-effetto tra attributi e oggetti.
- Sequence Analysis: vengono evidenziate delle relazioni tra oggetti su un periodo temporale.

Invece, tra gli algoritmi predittivi troviamo:

- Classification: in esso viene predetta l'appartenenza di un oggetto ad una classe.
- Regression: grazie a questo meccanismo, il valore di un attributo di un oggetto viene predetto.
- Time Series Analysis: il loro funzionamento è paragonabile a quello delle regressioni ma viene tenuto in considerazione anche il fattore temporale.
- Prediction: in esso viene fatta una stima del valore futuro.

3.3 Sequence Analysis

La Sequence Analysis è una famiglia di tecniche utilizzate per quantificare le distanze tra serie temporali categoriali. In particolare, questi algoritmi consentono di riepilogare sequenze o episodi frequenti nei dati. L'approccio della Sequence Analysis è completamente non parametrico e si divide in tre fasi.

L'output standard della prima fase della Sequence Analysis è una matrice di dissimilarità. Nella seconda fase, invece, le matrici di dissimilarità vengono utilizzate come input nelle tecniche di data reduction, principalmente rappresentate o da analisi di cluster o da multidimensional scaling. I gruppi ottenuti tramite data reduction possono poi essere eventualmente utilizzati in una terza fase per analisi successive.

La sequence analysis può essere applicata in diversi contesti come ad esempio: [28] [29]

- Bioinformatica: la Sequence Analysis è il processo utilizzato per sequenziare DNA, RNA e peptide per comprenderne le loro caratteristiche e la loro struttura. Una collezione di sequenze non costituisce di per sé nuovo contenuto informativo in grado di accrescere la conoscenza degli scienziati, se però viene confrontata questa nuova sequenza con quelle già note è possibile trarne conclusioni utili al progresso scientifico. In particolare, nel caso del DNA può essere usata per studiare geni e proteine mediante l'analisi delle somiglianze tra le sequenze confrontate.
- Chimica: in chimica la Sequence Analysis viene utilizzata per determinare ad esempio la sequenza di un polimero.
- Geo-Fisica: in questo ambito vengono studiate le relazioni tra una sequenza di punti dati o una sequenza di segnali al fine di determinare le proprietà fisiche della terra.
- Marketing: nel marketing viene invece usata nelle applicazioni per la gestione delle relazioni con i clienti, come ad esempio nei modelli NPTB.
- Sociologia: in sociologia viene applicata per studiare i modelli di sviluppo sociale, il corso della vita e l'andamento delle carriere lavorative.

3.4 Metodi di Sequence Classification

La Sequence Classification è un'importante attività di Data Mining in molte applicazioni del mondo reale. Negli ultimi decenni, molti metodi di classificazione delle sequenze sono stati proposti ed, in particolare, il metodo pattern-based è uno dei metodi di classificazione di sequenze più importanti. [33]

E' possibile suddividere i metodi di Sequence Classification in tre grandi categorie:

- Feature Based Classification
- Sequence Distance Based Classification
- Model Based Classification

3.4.1 Feature Based Classification

I metodi di Feature Based Sequence Classification possono essere suddivisi a loro volta in cinque categorie. Ora si andranno a vedere nel dettaglio e verranno analizzati pro e contro di ciascun metodo: [33]

- Explicit Subsequence Representation without Selection: inizialmente, l'approccio utilizzato nel trattare le sequenze discrete consisteva nel trattare ogni singolo elemento come una feature. Tuttavia, usando questo metodo si andavano a perdere le informazioni sull'ordine tra i diversi elementi e dunque la classificazione non era in grado di individuarne la sequenza.

  Per ovviare a questo problema, si è pensato di inserire dei segmenti di breve sequenza di k elementi consecutivi chiamati k-grammi. Dato un insieme di k-grammi, una sequenza può essere rappresentata o come un vettore che identifica la presenza o l'assenza dei k-grammi o come identificatore delle frequenze dei k-grammi. In questo metodo di rappresentazione delle caratteristiche, tutti i k-grammi (per un valore k specificato) vengono utilizzati esplicitamente come feature senza però avere feature selection.

- Explicit Subsequence Representation with Selection (Classifier-Independent): in questo approccio si utilizza un metodo di classificazione in cui un sequential pattern viene scelto come feature. I pattern che vengono definiti come feature solitamente sono sottoposti a molti vincoli e devono soddisfare diversi criteri: essere frequenti, essere in grado di identificare almeno una classe e non essere

ridondanti. Inoltre, questi modelli sono indipendenti dal classificatore in quanto qualsiasi classificatore progettato per dati vettoriali può essere applicato su dei dati che sono stati generati da modelli di questo tipo.

- Explicit Subsequence Representation with Selection (Classifier-Dependent): solitamente i metodi pattern-based sono universali e classificatori indipendenti. Tuttavia, in taluni casi è possibile filtrare alcuni modelli durante il processo di selezione per renderli classificatori dipendenti. Ad esempio, ciò è realizzabile tramite una tecnica di "salita del gradiente" in modo da apprendere la funzione di regressione logistica nello spazio di tutti i k-grammi. Il metodo sfrutta la struttura intrinseca dello spazio delle caratteristiche dei k-grammi per fornire automaticamente un insieme compatto di caratteristiche di quelli altamente discriminanti. L'idea chiave è quella di utilizzare una strategia di discesa delle coordinate limitata al gradiente per recuperare rapidamente le caratteristiche senza enumerare esplicitamente tutte le potenziali sottosequenze.

- Implicit Subsequence Representation: contrariamente alla rappresentazione esplicita della sottosequenza, questi metodi definiti "Kernel" impiegano, invece, una strategia di rappresentazione implicita della sottosequenza. Una funzione Kernel è l'ingrediente chiave per l'apprendimento con le Support Vector Machines (SVM) e definisce implicitamente uno spazio di funzionalità ad alta dimensionalità. Alcune funzioni del kernel $K(x,y)$ sono state presentate per misurare la somiglianza tra due sequenze x e y: una sequenza viene trasformata in un feature space e la funzione Kernel è il prodotto interno di due feature vectors trasformati. Il metodo SVM risulta essere un metodo efficace per la classificazione delle sequenze. In particolare, l'idea di base dell'applicazione di SVM sui dati di sequenza è mappare una sequenza in un feature space e trovare l'iperpiano con margine massimo per separare due classi. [34]

  A volte, non è necessario condurre esplicitamente la feature selection: date due sequenze, x,y, alcune funzioni del kernel, K(x,y), possono essere viste come la somiglianza tra due sequenze. Le sfide dell'applicazione di SVM alla

classificazione in sequenza includono la definizione di feature space e come velocizzare il calcolo delle matrici del kernel.

Uno dei kernel ampiamente utilizzati per la classificazione delle sequenze è il kernel "K-spectrum" o il "kernel stringa", che trasforma una sequenza in un feature vector. Ad esempio, il primo viene utilizzato per la classificazione delle proteine, mentre il secondo per la classificazione del testo.

Uno svantaggio dei metodi basati sul kernel è la sua difficile interpretazione e di conseguenza è estremamente complesso estrarre conoscenza. Per questo, nel corso degli anni sono state proposte delle soluzioni per rendere il modello più interpretabile come, ad esempio, quella di Sonnenburg [34]. Tramite l'utilizzo di un "kernel stringa" vengono assegnati dei pesi sulla base del valore di ciascuna caratteristica ed in questo modo gli utenti hanno un'idea più chiara riguardo l'importanza delle diverse funzionalità.

In generale, i "kernel stringa" o il "kernel K-spectrum" sono un metodo "feature based" ma nel corso del tempo sono state proposte diverse varianti che risultano essere, invece, "distance based". Questi sono stati applicati principalmente per la classificazione della sequenza proteica ma risultano avere dei limiti. Infatti, la distanza di allineamento locale può descrivere efficacemente la somiglianza tra due sequenze, non può essere utilizzata direttamente come funzione del kernel perché manca della proprietà di "positive definiteness". [34]

- Sequence Embedding: tutti i metodi sopra menzionati utilizzano le sottosequenze come features. In alternativa, il metodo di sequence embedding genera una rappresentazione vettoriale in cui ogni caratteristica non ha un'interpretazione chiara. La maggior parte degli approcci esistenti per questo metodo sono utilizzati per i testi nell'elaborazione del linguaggio naturale. Il principio alla base di questi metodi è che le parole che appaiono in contesti simili hanno significati simili. Ad esempio, il modello Word2vec utilizza una rete neurale a due strati per apprendere una rappresentazione vettoriale per ogni parola, mentre il modello Doc2vec estende il precedente apprendendo direttamente i feature vectors per intere frasi, paragrafi o documenti.

Recentemente, è stato proposto un ulteriore modello, Sqn2Vec, in grado di apprendere il sequence embedding prevedendo i suoi simboli singleton di appartenenza e pattern sequenziali (SP). L'obiettivo principale di Sqn2Vec è affrontare i limiti dei due approcci esistenti: i metodi basati su pattern spesso producono vettori di feature sparsi e ad alta dimensione, mentre i metodi di sequence embedding nell'elaborazione del linguaggio naturale possono fallire su set di dati con un vocabolario ridotto.

Valutando complessivamente tutti e cinque i metodi analizzati, si può dire innanzitutto che l'utilizzo di k-grammi come feature senza feature selection è semplice ed efficace nella pratica ma la lunghezza della caratteristica k non può essere grande e possono essere incluse molte caratteristiche ridondanti.

In secondo luogo, nel pattern-based method, la lunghezza di una feature non è limitata ma deve soddisfare determinati vincoli. Tuttavia, non è un compito banale estrarre in modo efficiente modelli di questo tipo.

In terzo luogo, i metodi di classificazione delle sequenze basati sulla selezione delle caratteristiche adattive possono invece selezionare automaticamente le caratteristiche dall'insieme di tutte le sottosequenze. L'idea di base è integrare la selezione delle features e la costruzione del classificatore nella stessa procedura. Dunque, questi metodi sono dipendenti dal classificatore, ovvero ogni algoritmo è applicabile solo a un classificatore specifico.

Invece, i metodi basati sul kernel possono mappare implicitamente la sequenza in uno spazio di caratteristiche ad alta dimensione senza estrazione esplicita delle features. La sfida in questo caso, consiste invece nello scegliere una corretta funzione del kernel e gestire in modo efficiente grandi insiemi di dati.

Infine, i metodi di sequence embedding generano una nuova rappresentazione vettoriale per ciascuna sequenza che può ottenere una migliore accuracy. Sfortunatamente, l'interpretazione semantica di ogni feature è molto complessa.

3.4.2 Sequence Distance Based Classification

I metodi Sequence Distance Based definiscono una funzione di distanza per misurare la somiglianza tra una coppia di sequenze. Una volta ottenuta una tale funzione di distanza, possiamo utilizzare alcuni metodi di classificazione esistenti, come K Nearest Neighbor Classifier (KNN) e SVM con kernel di allineamento locale per la classificazione della sequenza. In particolare, KNN dato un set di dati di sequenza etichettato T, un intero positivo k e una nuova sequenza s da classificare, il classificatore KNN trova i k vicini più prossimi di s in T e restituisce l'etichetta della classe dominante in kN N (s) come etichetta di s. [34]

La scelta delle misure di distanza è fondamentale per le prestazioni dei classificatori KNN. Per una semplice classificazione delle serie temporali, la distanza euclidea è un'opzione ampiamente adottata. Per due serie temporali s e s′, la distanza euclidea è:

$$\text{dist(s,s')} = \sqrt{\sum_{i=1}^{L} (s[i] - s'[i])^2}$$

La distanza euclidea di solito richiede che due serie temporali abbiano la stessa lunghezza e quando si applica il classificatore 1NN alle serie temporali, la distanza euclidea è sorprendentemente competitiva in termini di accuratezza, rispetto ad altre misure di similarità più complesse.

La distanza euclidea è però sensibile alle distorsioni nella dimensione temporale. Per superare questo problema, è stata introdotta la Dynamic Time Warping Distance (DTW) in quanto non richiede che due serie temporali abbiano la stessa lunghezza. Tuttavia, recenti risultati empirici suggeriscono fortemente che su grandi insiemi di dati, l'accuratezza delle misure elastiche converga con la distanza euclidea. Inoltre DTW viene solitamente calcolata dalla programmazione dinamica e avendo una complessità temporale quadratica risulta molto costosa su un set di dati di grandi dimensioni.

Per le sequenze simboliche, come le sequenze proteiche e le sequenze di DNA, sono comunemente adottate distanze basate sull'allineamento. Data una matrice di similarità e una penalità di gap, l'algoritmo di Needleman-Wunsch calcola un punteggio di allineamento globale ottimale tra due sequenze attraverso la programmazione dinamica. Contrariamente agli algoritmi di allineamento globale, gli algoritmi di allineamento locale,

come l'algoritmo di Smith-Waterman e BLAST, misurano la somiglianza tra due sequenze considerando le regioni più simili.

3.4.3 Model Based Classification

Una categoria di metodi di classificazione delle sequenze si basa su modelli generativi, che presuppongono che le sequenze in una classe siano generate da un sotto-modello M. Infatti, data una classe di sequenze, M modella la distribuzione di probabilità delle sequenze nella classe.

Solitamente si definisce un modello sulla base di alcune assunzioni e le distribuzioni di probabilità sono descritte da un insieme di parametri. In particolare, nella fase di addestramento vengono appresi i parametri di M e nella fase di classificazione viene assegnata una nuova sequenza alla classe con la probabilità più alta.

Il modello generativo più semplice è il classificatore di sequenze di Naive Bayes. Esso presuppone che, data una classe, le caratteristiche nelle sequenze siano indipendenti l'una dall'altra. E' considerato un modello relativamente semplice e per questo è stato ampiamente utilizzato dalla classificazione del testo e delle sequenze genomiche. Tuttavia, questo modello presenta un presupposto di indipendenza che nella pratica è spesso violato.

Altri due modelli di questo tipo sono Markov Model e Hidden Markov Model ed essi sono in grado di modellare la dipendenza tra elementi nelle sequenze. In particolare, Yakhnenko in passato ha applicato un modello di Markov di ordine k per classificare i dati di sequenze proteiche e di testo. Nel processo di addestramento, il modello viene addestrato in un setting discriminativo invece del setting generativo convenzionale in modo da aumentare il potere di classificazione dei metodi basati sul modello generativo. [34]

Diverso dal modello di Markov, il modello di Markov Nascosto presuppone che il sistema da modellare sia un processo di Markov con stati non osservati. Srivastava, in passato, ha utilizzato un profilo HMM per classificare le sequenze biologiche.

Un profilo HMM di solito ha tre tipi di stati: inserimento, corrispondenza ed eliminazione. Per apprendere le probabilità di transizione tra gli stati, vengono utilizzati esempi di training allineati. Per ogni classe viene appreso un profilo HMM ed esso rappresenta il profilo del set di dati di addestramento. Nella fase di classificazione, mediante

programmazione dinamica, in ciascuna classe una sequenza sconosciuta viene allineata con il profilo HMM. In particolare, la sequenza sconosciuta sarà classificata nella classe che presenta il punteggio di allineamento più elevato. [34]

Capitolo 4: Classificazione di sequenze per la previsione del genere

4.1 Introduzione alla classificazione

Nel machine learning la classificazione (classification) è un problema che consiste nell'identificare a quale categoria appartiene un elemento in input, sulla base di un modello di classificazione ottenuto in apprendimento automatico.

Ogni oggetto/elemento che viene osservato è detto istanza e l'algoritmo che implementa la classificazione è detto classificatore (classifier). Il classificatore esamina l'istanza e la etichetta con una classe. Ad esempio, un esempio semplice di contesto in cui viene applicata la classificazione è l'identificazione delle email come spam/no spam. In questo caso, la classificazione è detta binomiale o binaria. Al contrario, se le classi sono più di due, si parla di classificazione multiclasse. Un altro modello tipico in cui viene utilizzata una classificazione è la sentiment analysis, in cui

La classificazione può essere di tipo supervised o unsupervised. Nel caso specifico, in questo studio verrà utilizzato un approccio supervised. Nell'apprendimento supervisionato, gli algoritmi apprendono dai dati etichettati. Dopo aver compreso i dati, l'algoritmo determina quale etichetta deve essere data ai nuovi dati associando i pattern ai nuovi dati non etichettati.

In quella unsupervised, invece, i due passaggi principali sono la generazione di cluster e l'assegnazione di classi. Il passaggio successivo prevede l'assegnazione manuale della classe a ogni cluster. Nella *figura 8* è possibile osservare l'intero processo:

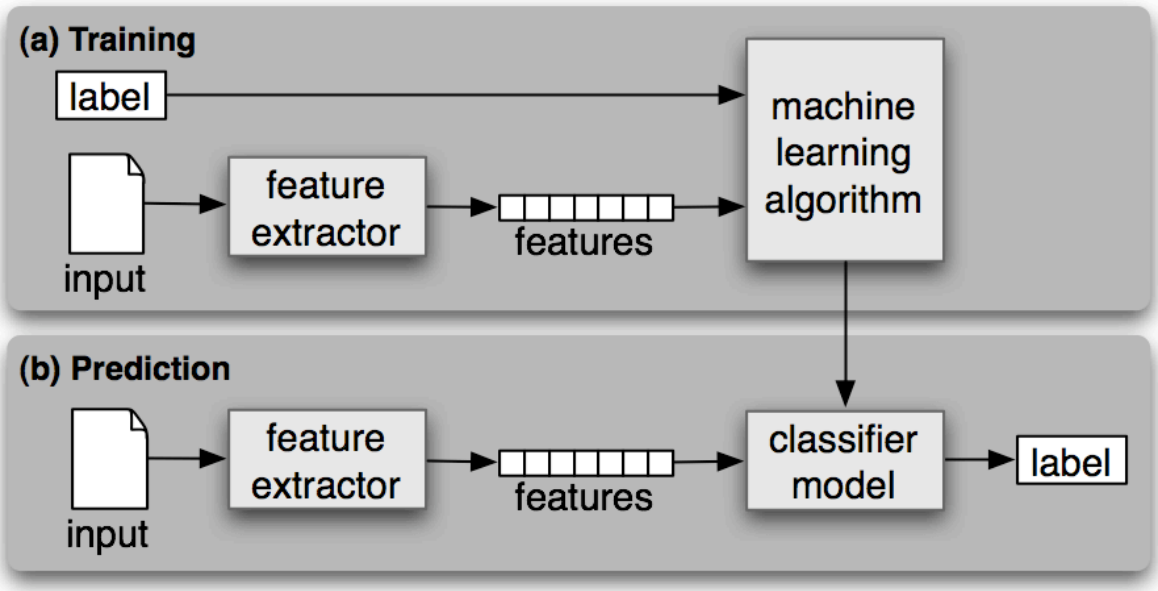

Figura 6 – Classificazione

4.2 Creazione del dataset

Per realizzare le analisi, successivamente ad un'approfondita ricerca sull'argomento, si è partiti dalla creazione del Dataset. L'idea di partenza era quella di realizzare un dataset composto da circa 1000 Movie Clip caratterizzato da diverse informazioni.

Le prime informazioni che sono state reperite sono state il titolo della clip in questione ed il relativo link di Youtube. Principalmente, i canali su cui sono state trovate le clip sono stati "*Movieclips*" e "*Movieclips Coming Soon*". Riporto ad esempio una clip utilizzata:

Link	**Titolo**
https://www.youtube.com/watch?v=xTvdOjsJXRY&t=1s	Robin Hood

Il dataset creato contiene 938 MovieClip. Su di esso, poi, si è proceduto con le analisi. Successivamente, infatti, è stato applicato un algoritmo che prende in input un url di un video da Youtube, divide il video in inquadrature e salva un frame per inquadratura.

Per velocizzare il processo, l'algoritmo è stato implementato affinchè prendesse autonomamente da un file excel tutti i link con i relativi titoli del dataset, uno alla volta, in modo da non dover rilanciare la procedura per un numero di volte pari al numero di componenti del dataset. Di seguito riporto il codice in questione realizzato in collaborazione con Simone Magnafico e Bartolomeo Vacchetti:

```
import pandas as pd
import numpy as np

# Standard PySceneDetect imports:
from scenedetect import VideoManager
from scenedetect import SceneManager

# For content-aware scene detection:
from scenedetect.detectors import ContentDetector
import cv2
import os
import shutil
import pafy
```

In questa prima parte vengono importate le librerie necessarie per il codice.

```
def find_scenes(video_path, threshold=30.0):
    # Create our video & scene managers, then add the detector.
    video_manager = VideoManager([video_path])
    scene_manager = SceneManager()
    scene_manager.add_detector(
        ContentDetector(threshold=threshold))

    # Improve processing speed by downscaling before processing.
    video_manager.set_downscale_factor()

    # Start the video manager and perform the scene detection.
    video_manager.start()
    scene_manager.detect_scenes(frame_source=video_manager)

    # Each returned scene is a tuple of the (start, end) timecode.
    return scene_manager.get_scene_list()
```

In questa seconda parte, viene creato lo scene manager e viene aggiunto il detector che permetterà di individuare le scene.

```
frame_start=[]
frame_end=[]
frame_inter=[]

df_test=pd.read_excel('/Users/letiziademarchi/Desktop/ElencoDataset.xlsx')

scarti='/Users/letiziademarchi/Desktop/scarti'
dest='/Users/letiziademarchi/Desktop/Video_frame'
current_frame=0
```

Nella terza parte viene utilizzata la libreria pandas, per caricare il file "*ElencoDataset*" il quale contiene le informazioni sopra citate, ovvero il link del video ed il relativo titolo.

Inoltre, vengono create le due cartelle "*scarti*" e "*Video_frame*". Nella seconda cartella verranno inseriti tutti i frame individuati dal codice.

```
for i in range(0,len(df_test)):
   url=df_test.iloc[i,0]
   title=df_test.iloc[i,1]

   #for Name in df_test['Name']:
   vPafy=pafy.new(url)
   play=vPafy.getbestvideo(preftype='webm')
   scenes = find_scenes(play.url)

   for scene in scenes:
      frame_start.append(scene[0].frame_num)
      frame_end.append(scene[1].frame_num)
      if (scene[0].frame_num+scene[1].frame_num)%2==0:
         frame_inter.append(int((scene[0].frame_num+scene[1].frame_num)/2))
      else:
```

```
        frame_inter.append(int((scene[0].frame_num+scene[1].frame_num+1)/2))


df_start=pd.DataFrame(frame_start, columns=['Start Frame'])
df_end=pd.DataFrame(frame_end, columns=['End Frame'])
df_inter=pd.DataFrame(frame_inter, columns=['Inter. Frame'])
df_frame=pd.concat([df_start, df_end,df_inter],axis=1)

#print(df_frame)
video=cv2.VideoCapture(play.url)

newpath = r'/Users/simonemagnafico/Desktop/Video_frame/{}'.format(title)

try:

# creating a folder named data
    if not os.path.exists(newpath):
        os.makedirs(newpath)

# if not created then raise error
except OSError:
    print ('Error: Creating directory of data')


while(True):
    ret, frame=video.read()
#cv2.imshow('frame',frame)

    #current_frame=0

    if ret:
        if current_frame in df_frame['Inter. Frame'].values:
            name=title+str(current_frame)+'.jpg'
```

```
            print("Creating..." + name)

            cv2.imwrite(name, frame)

            shutil.move(name, newpath)
            current_frame+=1
        else:
            current_frame+=1

    else:
        break

video.release()
cv2.destroyAllWindows()
```

Nell'ultima parte il codice crea nella cartella sopra definita una cartella per ogni singola clip del dataset e all'interno di essa inserisce un frame per ogni inquadratura individuata. Nella *figura 9* viene riportato un esempio di frame individuati grazie al codice sopra riportato.

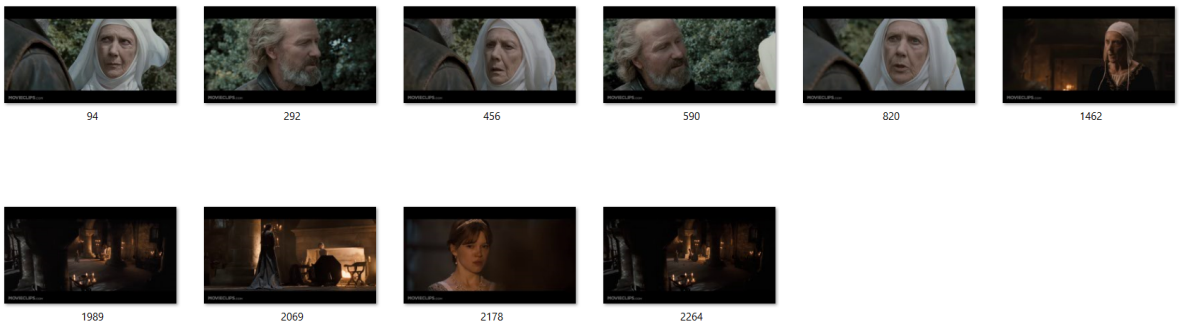

Figura 7 - Frame individuati di Robin Hood

Una volta ottenute 938 cartelle pari al numero di Movie Clip presenti nel dataset, su ognuna di esse è stato applicato un codice in grado di tradurre ogni frame nel tipo di inquadratura corrispondente. I tipi di inquadratura presi in considerazione sono 8:

- 0=Long Shot (LS)= Campo Lungo (CL)
- 1=Medium Shot (MS)= Campo Medio (MS)
- 2=Full Figure (FF)= Figura Intera (FI)
- 3=American Shot (AS)= Piano Americano (PA)
- 4=Half Figure (HF)= Mezza Figura (MF)
- 5=Half Torso (HT)= Mezzo Busto (MB)
- 6= Close Up (CU)= Primo Piano (PP)
- 7=Extreme Close UP (ECU)= Primissimo Piano (PPP)

Nel caso dell'esempio preso in considerazione, ovvero Robin Hood, la sequenza di inquadrature tradotta da immagine a numero è: [6 6 6 6 6 5 6 5 6 6], ovvero si alternano primi piani (individuati con il numero 6) a mezzi busti (individuati con il numero 5).

Questo procedimento è stato applicato a tutti i 938 componenti del dataset e, pertanto, al Dataset oltre a "*Link*" e "*Titolo*" è stata aggiunta l'informazione della "*Sequence*".

Link	**Titolo**	**Sequence**
https://www.youtube.com/watch?v=xTvdOjsJXRY&t=1s	Robin Hood	6 6 6 6 6 5 6 5 6 6

Successivamente, utilizzando il sito IMDB, è stato inserito il genere per ciascuna delle clip prese in considerazione. Per uniformità, si è deciso di categorizzare i film in quattro generi:

- Action
- Comedy
- Drama
- Horror

Ad esempio, per Robin Hood è stato definito il genere "*Action*".

Link	**Titolo**	**Sequence**	**Genere**

https://www.youtube.com/watch?	Robin Hood	6 6 6 6 6 5 6 5 6 6	Action

Infine, per caratterizzare ulteriormente il dataset, sono state inserite come informazioni aggiuntive:

- "End Frame": identifica l'ultimo frame della clip
- "Lunghezza": identifica il numero degli elementi della sequenza
- "Lunghezza_media": identifica la lunghezza media dei frame in quella determinata sequenza.

4.3 Caratterizzazione del dataset

Come già anticipato, il dataset è composto da 938 elementi. Per uniformità di analisi, le clip dei film selezionati sono state prese dividendo equamente per le lettere alfabetiche:

- A-C: 196 records
- D-I: 194 records
- J-P: 272 records
- R-Z: 276 records

Per quanto riguarda i generi, come si può vedere dal grafico sotto riportato (*figura 10*), la quantità di clip identificate con i generi Action, Comedy o Drama all'incirca si equivalgono. Al contrario, invece, nel genere Horror sono state identificate molte meno clip. In particolare, abbiamo la seguente numerosità:

- Action: 279 clip
- Comedy: 294 clip
- Drama: 282 clip
- Horror: 83 clip

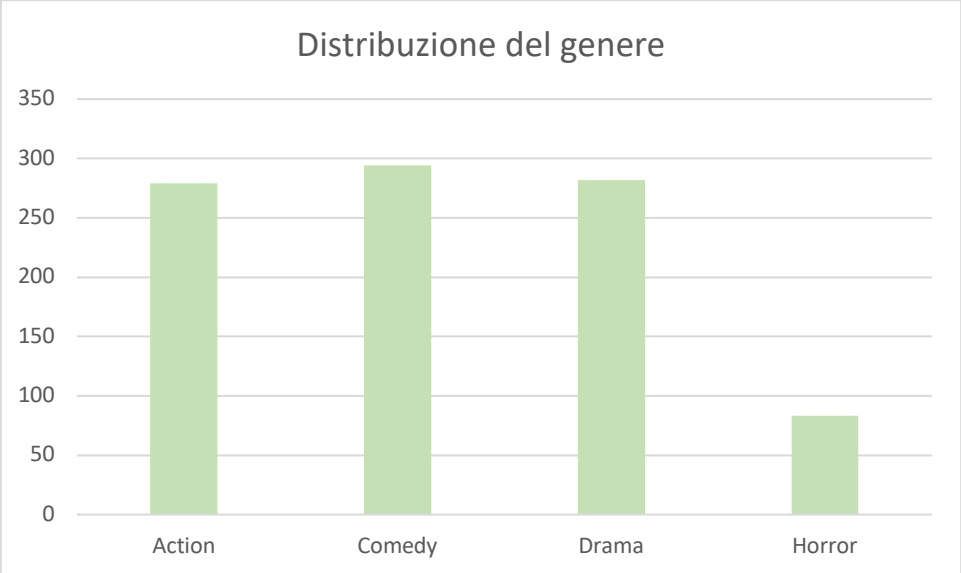

Figura 8 - Distribuzione del genere

Invece, per quanto riguarda la distribuzione della lunghezza delle sequenze, si può osservare nella *figura 11* che la lunghezza più diffusa tra le sequenze dal dataset si attesti in un valore compreso tra 12 e 22. Dal grafico, inoltre, si può notare che la maggior parte delle clip sono caratterizzate da sequenze tra i 2 e i 52 frame. Infatti, la lunghezza media delle sequenze presenti nel dataset è di 33 elementi.

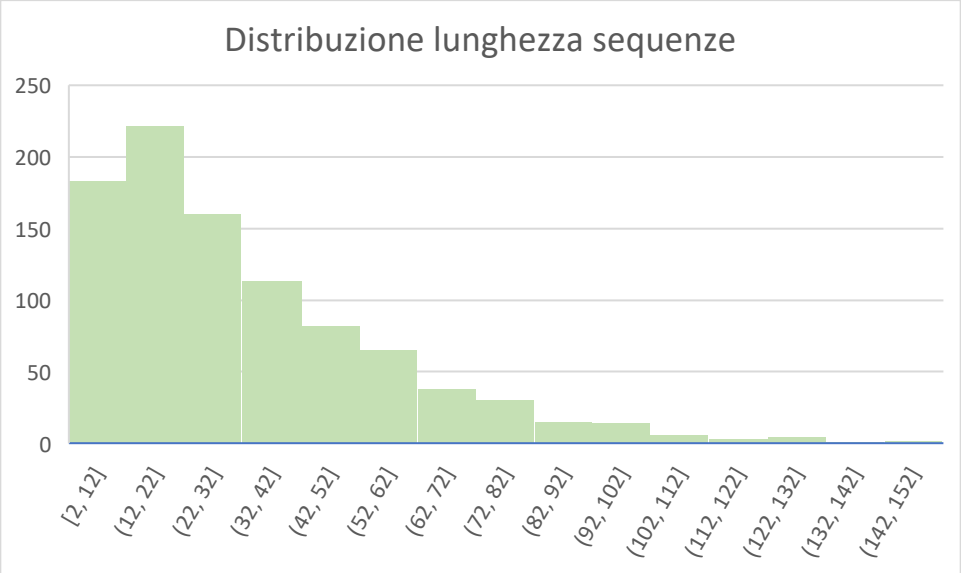

Figura 9 - Distribuzione della lunghezza delle sequenze

Analizzando i tipi di inquadrature presenti nel dataset, si osserva nella *figura 12* che il tipo di inquadratura maggiormente presente è il mezzo busto, a seguire il primo piano e la figura intera. Il tipo, invece, presente in maniera minore nel dataset è il primissimo piano.

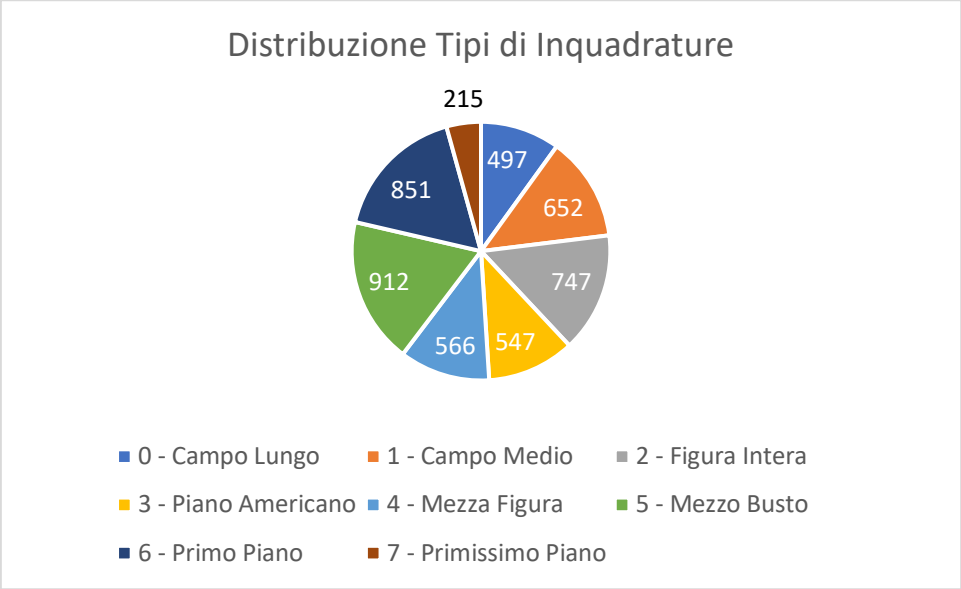

Figura 10 - Distribuzione Tipi di Inquadrature

4.4 Obiettivo della classificazione

Dopo un'attenta analisi del dataset, la prima analisi che si è svolta è stata quella sull'identificazione del genere. La domanda che ci si è posti è stata: "*può un algoritmo, essere in grado di riconoscere il genere cinematografico della clip in questione note le sequenze delle tipologie di inquadrature*?".

Per fare ciò, si è scelto di affidarsi ad un approccio supervised, ovvero all'algoritmo viene definita dal principio un'etichetta ed è essa che si deve predire.

Per chiarire, si riporta un esempio sempre relativo alla clip presa in considerazione di Robin Hood:

Sequenza = [6, 6, 6, 6, 6, 5, 6, 5, 6, 6]

Genere = Action

In questo caso, il codice conoscendo a priori la sequenza di inquadrature, in questo caso composta da primi piani e mezzi busti, dovrebbe essere in grado di prevedere il genere di quest'ultimo, ovvero *Action*.

Capitolo 5: Implementazione e analisi

5.1 Scelta del modello

Per fare ciò, si è pensato all'utilizzo delle reti LSTM. Esse infatti, come già detto, sono particolarmente adatte per la risoluzione di problemi molto complessi in quanto possono elaborare sia singoli dati che una sequenza, come ad esempio un video completo. Inoltre, esse aiutano ad evitare problemi legati alla dipendenza a lungo termine.

Il processo di trasferimento dei dati è lo stesso delle reti neurali ricorrenti standard ma tuttavia, l'operazione di propagazione delle informazioni è diversa. Quando l'informazione arriva in input, la rete decide quali informazioni elaborare ulteriormente e quali informazioni, invece, lasciare andare.

L'operazione principale è costituita da celle e cancelli: le celle sono la memoria, mentre i cancelli indicano quali dati sono utili da conservare e quali non lo sono, rendendo possibile il lancio. In questo modo solo i dati rilevanti passano attraverso la catena di sequenza per utile per la previsione.

5.2 Il modello

Di seguito viene riportato il modello utilizzato per quest'analisi. Inizialmente per testare i parametri più corretti da utilizzare non è stato utilizzato nessun tipo di Cross Validation in quanto avrebbe solamente aumentato il tempo di elaborazione dell'algoritmo senza portare ad un miglioramento dei risultati.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.models import Sequential
```

```
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Bidirectional

from sklearn.model_selection import train_test_split

from keras.utils.np_utils import to_categorical

from keras.callbacks import EarlyStopping
```

Nella prima parte del codice vengono importate tutte le librerie che risulteranno utili per l'analisi successiva. In particolare, utilizzando in input un file .csv è stata importata la libreria pandas la quale risulta particolarmente adatta alla gestione dei dati su un file proveniente da Excel.

```
#read csv

df=pd.read_csv('/content/Dataset_completo.csv',squeeze=True, sep=";")

df.head()
```

Nella seconda parte viene importato il file Excel che verrà utilizzato come dataset. Tra le informazioni che contiene (già descritte nel capitolo precedente), in questa analisi verranno considerate solo quella sulla Sequence e quella sul Genere.

```
# The maximum number of words to be used
 MAX_NB_WORDS = 9
 # Max number of words in each complaint.
 MAX_SEQUENCE_LENGTH = 144
 # This is fixed.
 EMBEDDING_DIM = 32
 #Spit the sequences
 tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters=' ', lower=True)
 tokenizer.fit_on_texts(df['Sequence'])
 word_index = tokenizer.word_index
 print('Found %s unique tokens.' % len(word_index))
 #Definition of X
```

```
X = tokenizer.texts_to_sequences(df.Sequence.values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)

Output:
Found 8 unique tokens.
Shape of data tensor: (937, 144)
```

In questa terza parte viene definita la X, ovvero la sequenza da analizzare. Osservando l'output, vengono rilevati 8 tokens in quanto le tipologie di inquadrature considerate sono 8. Per quanto riguarda invece le dimensioni del data tensor, esse sono 937x144 in quanto 937 sono le righe del dataset, mentre 144 è la MAX_SEQUENCE_LENGTH. Essa è stata impostata su questo valore perché è la lunghezza della stringa più lunga presente nel dataset. Infatti, uno dei limiti che si affronterà meglio alla fine di questa analisi, è dovuto al fatto che le sequenze sono tutte di lunghezza differente, ma un modello di questo tipo necessita di un valore fisso per la dimensione, in modo da creare correttamente il data tensor.

Per splittare in modo corretto tutte le sequenze presenti nel file .csv è stata utilizzata la funzione "*pad_sequences*".

```
#Definition of Y
Y = pd.get_dummies(df.Genere.values)
print('Shape of label tensor:', Y.shape)
Output:
Shape of label tensor: (937, 4)
```

Successivamente ad X, è stata definita Y. Si noti come in questo caso le dimensioni del label tensor siano 937x4 in quanto i generi da predire considerati nell'analisi sono quattro.

```
#Definition of X_train, X_test, Y_train e Y_test

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state = 42)

print(X_train.shape,Y_train.shape)

print(X_test.shape,Y_test.shape)
```

Output:

```
(750, 144) (750, 4)

(188, 144) (188, 4)
```

In questo passaggio vengono definite le dimensioni della parte del dataset destinata al train del modello e di quella destinata al test. Nel caso in specifico, al dataset viene data un'ampiezza del 20% del totale del dataset e le dimensioni finali sono quelle riportate sopra.

```
#Definition of model

model = Sequential()

model.add(Embedding(MAX_NB_WORDS,EMBEDDING_DIM,input_length=X.shape[1]))

model.add(LSTM(128, kernel_initializer="glorot_uniform"))

model.add(Dense(4, activation="softmax"))

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

print(model.summary())
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 144, 32)           288

 lstm (LSTM)                 (None, 128)               82432

 dense (Dense)               (None, 4)                 516
```

```
=================================================================
Total params: 83,236
Trainable params: 83,236
Non-trainable params: 0
_________________________________________________________________
None
```

Il modello definito è così composto:

- Embedding: esso viene aggiunto in quanto permette di trattare le sequenze sopra definite. In particolare, MAX_NB_WORDS è la dimensione del vocabolario nei dati di testo. Ad esempio, se i dati sono codificati in numeri interi con valori compresi tra 0 e 10, la dimensione del vocabolario sarà di 11 parole. Nel caso specifico è pari a 9 in quanto i valori sono compresi tra 0 e 8. EMBEDDING_DIM è la dimensione dello spazio vettoriale in cui verranno incorporate le parole. Definisce la dimensione dei vettori di output da questo livello per ogni parola. Nel caso in esame, dopo aver testato diversi parametri si è optato per 32. Input_length è la lunghezza delle sequenze di input ed infatti viene presa direttamente dalla shape[0] di X.
- LSTM: dopo aver effettuato numerosissime prove sul numero di livelli LSTM da inserire, è stato deciso 128. Inizialmente non era stato inserito il kernel initializer, ma a causa di alcuni problemi di overfitting (che vedremo a breve) generatosi si è optato per il suo inserimento. Esso serve per definire i pesi dei livelli di input forniti.

```
#Fit of the model

history = model.fit(X_train, Y_train, epochs=50, batch_size=10, validation_split=0.1,callbacks=[EarlyStopping(monitor='val_loss',patience=15, min_delta=0.0001)])
```

Output:

```
Epoch 1/50
68/68 [==============================] - 12s 143ms/step - loss: 0.5695 - accuracy: 0.3111 - val_loss: 0.5293 - val_accuracy: 0.3467
Epoch 2/50
68/68 [==============================] - 9s 134ms/step - loss: 0.5405 - accuracy: 0.3230 - val_loss: 0.5267 - val_accuracy: 0.3733
Epoch 3/50
```

```
68/68 [==============================] - 9s 133ms/step - loss:
0.5323 - accuracy: 0.3822 - val_loss: 0.4935 - val_accuracy: 0.4667
Epoch 4/50
68/68 [==============================] - 9s 133ms/step - loss:
0.5223 - accuracy: 0.3985 - val_loss: 0.5032 - val_accuracy: 0.3867
Epoch 5/50
68/68 [==============================] - 9s 134ms/step - loss:
0.5192 - accuracy: 0.4059 - val_loss: 0.4978 - val_accuracy: 0.4267
Epoch 6/50
68/68 [==============================] - 9s 133ms/step - loss:
0.5168 - accuracy: 0.3956 - val_loss: 0.5038 - val_accuracy: 0.4533
Epoch 7/50
68/68 [==============================] - 9s 134ms/step - loss:
0.5160 - accuracy: 0.4237 - val_loss: 0.4949 - val_accuracy: 0.4533
Epoch 8/50
68/68 [==============================] - 9s 133ms/step - loss:
0.5093 - accuracy: 0.4341 - val_loss: 0.5003 - val_accuracy: 0.4000
Epoch 9/50
68/68 [==============================] - 9s 132ms/step - loss:
0.5113 - accuracy: 0.4015 - val_loss: 0.5013 - val_accuracy: 0.4533
Epoch 10/50
68/68 [==============================] - 9s 133ms/step - loss:
0.5068 - accuracy: 0.4415 - val_loss: 0.4960 - val_accuracy: 0.3867
Epoch 40/50
68/68 [==============================] - 9s 134ms/step - loss:
0.3132 - accuracy: 0.7289 - val_loss: 0.5986 - val_accuracy: 0.4133
Epoch 41/50
68/68 [==============================] - 9s 136ms/step - loss:
0.3167 - accuracy: 0.7393 - val_loss: 0.5719 - val_accuracy: 0.4133
Epoch 42/50
68/68 [==============================] - 9s 135ms/step - loss:
0.2889 - accuracy: 0.7793 - val_loss: 0.5834 - val_accuracy: 0.4800
Epoch 43/50
68/68 [==============================] - 9s 135ms/step - loss:
0.2558 - accuracy: 0.7926 - val_loss: 0.6073 - val_accuracy: 0.4400
Epoch 44/50
68/68 [==============================] - 9s 136ms/step - loss:
0.2281 - accuracy: 0.8296 - val_loss: 0.7021 - val_accuracy: 0.4400
Epoch 45/50
68/68 [==============================] - 9s 134ms/step - loss:
0.2199 - accuracy: 0.8341 - val_loss: 0.6824 - val_accuracy: 0.4400
Epoch 46/50
68/68 [==============================] - 9s 135ms/step - loss:
0.2165 - accuracy: 0.8459 - val_loss: 0.6759 - val_accuracy: 0.4267
Epoch 47/50
68/68 [==============================] - 9s 134ms/step - loss:
0.1887 - accuracy: 0.8770 - val_loss: 0.6580 - val_accuracy: 0.4800
Epoch 48/50
68/68 [==============================] - 9s 134ms/step - loss:
0.1767 - accuracy: 0.8889 - val_loss: 0.7232 - val_accuracy: 0.4667
Epoch 49/50
68/68 [==============================] - 9s 136ms/step - loss:
0.1657 - accuracy: 0.9037 - val_loss: 0.7349 - val_accuracy: 0.4400
Epoch 50/50
68/68 [==============================] - 9s 135ms/step - loss:
0.1521 - accuracy: 0.9022 - val_loss: 0.7721 - val_accuracy: 0.4267
```

Il modello è stato testato su 50 epoche e con un batch size pari a 5. Di sopra sono state riportate le prime 10 epoche e le ultime 10. Come si può osservare, si parte da un'accuratezza di train del 31% e si arriva alla fine dopo 50 epoche ad un'accuratezza di train del 90%.

```
#Evaluation of the model

accr = model.evaluate(X_test,Y_test)

#Preparation for Confusion Matrix

p=model.predict(X_test)

y_scores=p.argmax(1)

y_act=[]

Y_test_t=Y_test.T

for i in Y_test_t:

   n=Y_test_t[i].argmax()

   y_act.append(n)

print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

Output:

```
6/6 [==============================] - 0s 44ms/step - loss: 0.8368
- accuracy: 0.3617
Test set
  Loss: 0.837
  Accuracy: 0.362
```

5.2.3 Analisi dei risultati

Alla luce dei risultati ottenuti, il modello è stato analizzato. Per far ciò, si è scelto di utilizzare i due grafici sull'andamento del Loss e dell'Accuracy, la Confusion Matrix e il Classification Report.

```
#Loss visualization

plt.title('Loss')

plt.plot(history.history['loss'], label='train')
```

```
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
#Accuracy visualization
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show();
```

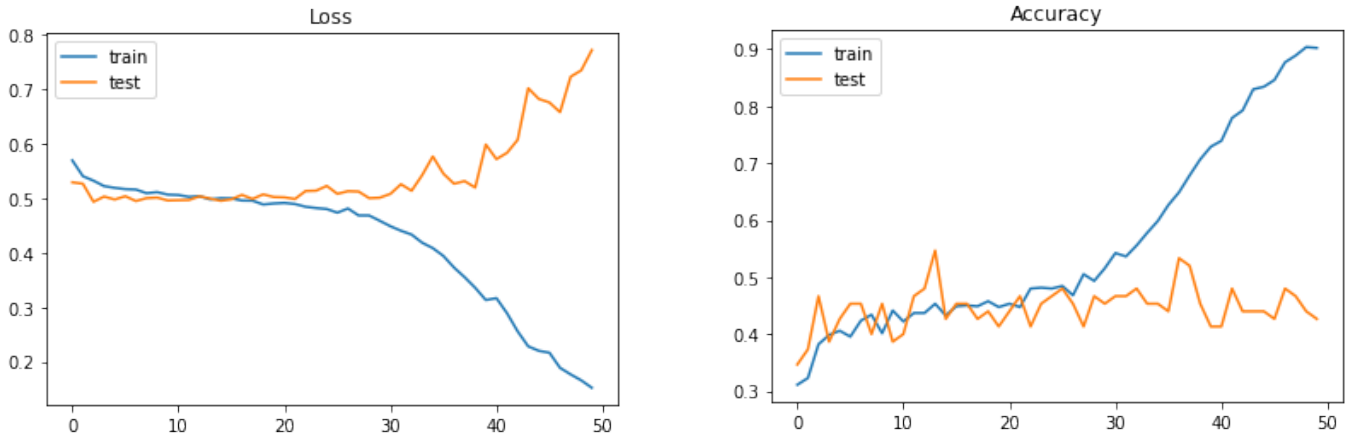

Figura 11 - Andamento Loss e Accuracy con LSTM=128

Osservando il grafico, per quanto riguarda i dati di train si può notare come scendano in maniera lineare per quanto riguarda il Loss, mentre salgano allo stesso modo per l'Accuracy. Da ciò si deduce che il modello, aumentando le epoche percepisce i dati di train correttamente.

Al contrario invece, i dati di validation-test presentano un andamento simile ai dati di train fino a circa 20 epoche, per poi, in un caso, crescere esponenzialmente, mentre nell'altro attestarsi in un valore compreso tra il 40% ed il 50%.

Complessivamente, infatti, il modello risulta avere un'accuracy di test del 36%, dunque poco soddisfacente ai fini degli obiettivi preposti.

Quello che accade è il fenomeno dell'overfittitng, cioè il sovradattamento durante il processo di apprendimento induttivo. Un modello è soggetto a overfitting quando funziona bene con i dati di addestramento, ma non con i dati di test, in quanto il modello

memorizza i dati che ha osservato, ma non è in grado di generalizzare il modello con dati non osservati.

I metodi generalmente utilizzati per ridurre l'overfitting in un modello sono diversi, ad esempio:

- Early Stopping: è una forma di regolarizzazione durante l'addestramento di un modello con un metodo iterativo. Poiché tutte le reti neurali imparano esclusivamente utilizzando la discesa del gradiente. Questo metodo aggiorna il modello in modo da adattarlo meglio ai dati di training ad ogni iterazione. Fino a un certo punto, ciò migliora le prestazioni del modello sui dati del set di test ma passato quel punto, il miglioramento dell'adattamento del modello ai dati di addestramento porta a un aumento dell'errore di generalizzazione. Le regole di arresto anticipato forniscono indicazioni su quante iterazioni possono essere eseguite prima che il modello inizi a essere sovradimensionato. Nel caso in esame, esso è stato inserito ma non ha portato alla diminuzione dell'overfitting.

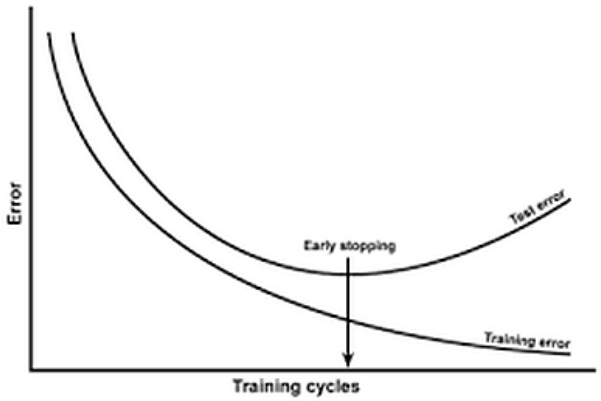

Figura 12 - Early Stopping

- Regularization: è una tecnica usata per ridurre la complessità del modello. Viene, infatti, aggiunto un fattore di penalità alla funzione loss. Le tecniche comunemente più utilizzate sono chiamate L1 e L2.

  La tecnica L1 mira a minimizzare il valore assoluto dei pesi, mentre la tecnica L2 mira a minimizzare la grandezza al quadrato dei pesi. Se i dati sono troppo complessi e non si riesce a modellarli correttamente, L2 risulta la regolarizzazione più adatta. Al contrario se i dati sono abbastanza semplici L1 dovrebbe performare meglio e di conseguenza limitare il rischio di overfitting.

L1 Regularization	L2 Regularization
1. L1 penalizes sum of absolute values of weights.	1. L2 penalizes sum of square values of weights.
2. L1 generates model that is simple and interpretable.	2. L2 regularization is able to learn complex data patterns.
3. L1 is robust to outliers.	3. L2 is not robust to outliers.

Figura 13 - Confronto L1 e L2

Nel caso in esame, sono stati testati entrambi i regularizer, anche su un numero maggiore di livelli LSTM ma nessuno dei due ha portato ad un miglioramento della condizione di overfitting.

- Dropouts: è una tecnica di regolarizzazione che impedisce l'overfitting delle reti neurali. I metodi di regolarizzazione come L1 e L2 riducono l'overfitting modificando la funzione di costo mentre il dropout modifica la rete stessa. Esso elimina casualmente i neuroni dalla rete neurale durante l'allenamento in ogni iterazione. Le diverse reti si sovrapporranno in modi diversi, quindi l'effetto netto del dropout sarà quello di ridurre l'overfitting.

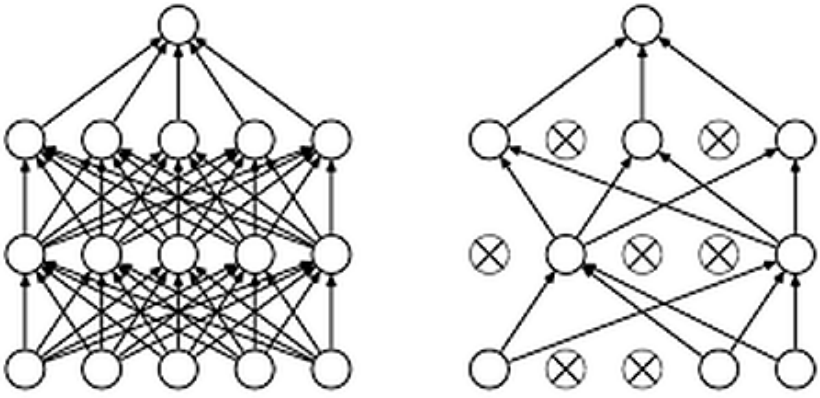

Figura 14 - Dropout

Come si vede nella *figura 16*, i dropout vengono utilizzati per rimuovere casualmente i neuroni durante il train della rete neurale.

Nel caso considerato, si è provato ad inserire su più casi il dropout ma in nessuno dei casi presi in esame ha portato ad un miglioramento dei risultati, in quanto il dataset in considerazione è troppo limitato.

- Semplificazione del modello: per diminuire la complessità, è possibile rimuovere i livelli o ridurre il numero di neuroni per rendere la rete più piccola.

Infatti, nelle prossime pagine verrà proposto un modello con un numero inferiore di livelli LSTM e verranno confrontate e analizzate le sue performance.

In ultimo, nel modello è stata inserita la Confusion Matrix e il Classification Report per analizzare i risultati nel dettaglio:

```
#Confusion Matrix e Classification Report
from sklearn.metrics import confusion_matrix, classification_report
from matplotlib import pyplot as plt
import seaborn as sns
def plot_confusion_matrix(y_act, y_scores, classNames):
   classes = len(classNames)
   cm = confusion_matrix(y_act, y_scores)
   print("**** Confusion Matrix ****")
   print(cm)
   print("**** Classification Report ****")
   print(classification_report(y_act, y_scores, target_names=classNames))
   con = np.zeros((classes,classes))
   for x in range(classes):
      for y in range(classes):
         con[x,y] = cm[x,y]/np.sum(cm[x,:])
   plt.figure(figsize=(40,40))
   sns.set(font_scale=3.0) # for label size
   df = sns.heatmap(con, annot=True,fmt='.2', cmap='Blues',xticklabels= classNames, yticklabels= classNames)
```

```
df.figure.savefig("image2.png")
```

```
classNames = ['Action', 'Comedy', 'Drama', 'Horror']
```

```
plot_confusion_matrix(y_act,y_scores, classNames)
```

Output:

```
**** Confusion Matrix ****
[[26 20  7  8]
 [18 25 13  3]
 [21 17 15  4]
 [ 3  5  1  2]]
```

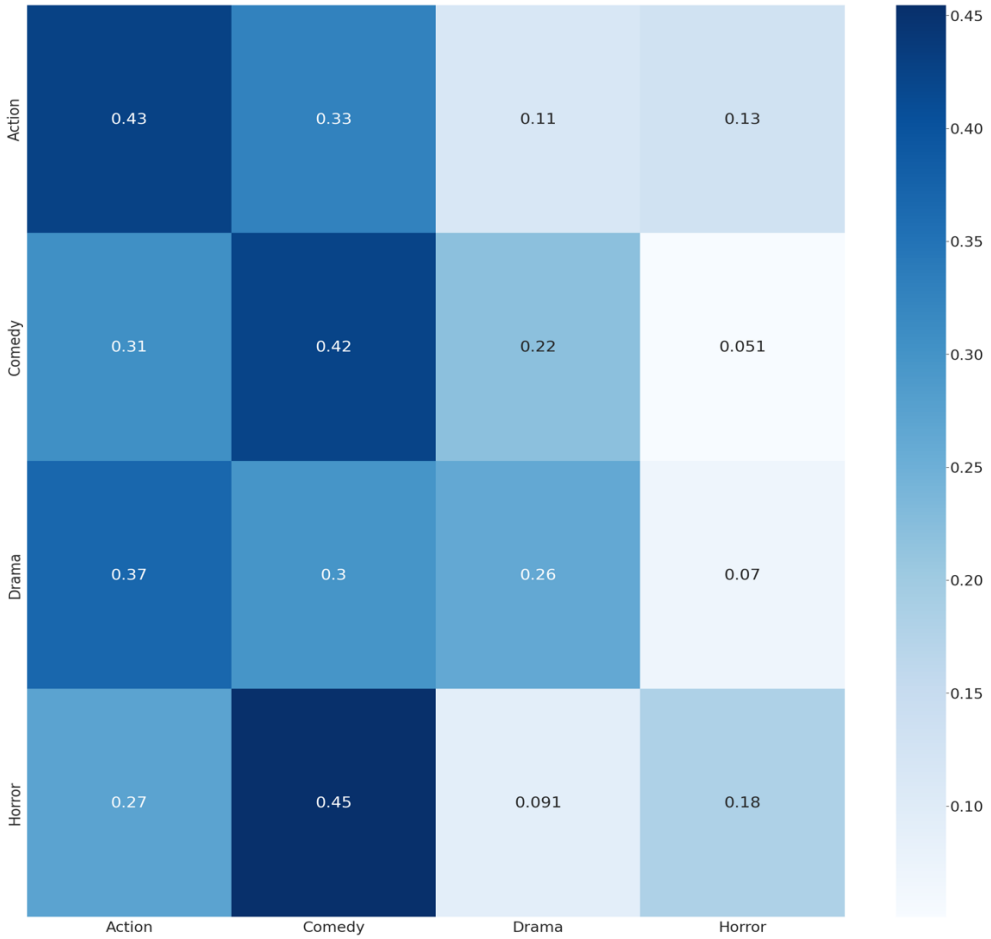

Figura 15 - Confusion Matrix con LSTM=128

La matrice di confusione nella *figura 17* permette di comprendere le performance di un modello predittivo di classificazione in modo da determinare quanto questo modello sia accurato ed efficace.

Per comprendere meglio l'incrocio tra valori effettivi e previsti alla matrice di confusione, si può semplificare il caso considerando il caso dell'email classificata come spam o meno. In questo caso la matrice sarà a due classi, ma i valori per la matrice in esame, che invece

è multiclasse, possono essere calcolati semplicemente sommando a seconda dei casi i valori sotto elencati:

- True positive (TP, o veri positivi): sono i casi in cui è stato previsto che la mail fosse spam, e lo sono realmente.
- True negative (TN, o veri negativi): sono i casi in cui è stato previsto che la mail non fosse spam e non lo è veramente.
- False positive (FP, o falsi positivi): sono i casi in cui è stato previsto che le mail fossero spam, ma in realtà non lo erano (conosciuti anche come errori di I tipo).
- False negative (FN, o falsi negativi): il modello ha previsto che la mail non è spam, anche se in realtà lo è (conosciuti anche come errori di II tipo).

Nel caso preso in esame, il modello, nel caso del genere Azione, lo predice correttamente il 43% delle volte, ma nel 33% dei casi scambia un film Comedy per Action. Il genere Comedy invece viene predetto correttamente nel 42% dei casi ma anche in questo caso il 31% delle volte un film Comedy viene scambiato per un Action. Un film Drama, viene invece riconosciuto correttamente il 26% dei casi, ma nel 37% delle volte viene identificato come Action. Per quanto riguarda gli Horror invece, i valori sono molto bassi in quanto il dataset è molto sbilanciato ed i record identificati con il genere Horror sono molti meno rispetto agli altri tre presenti nel dataset.

```
**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.38      0.43      0.40        61
      Comedy       0.37      0.42      0.40        59
       Drama       0.42      0.26      0.32        57
      Horror       0.12      0.18      0.14        11

    accuracy                           0.36       188
   macro avg       0.32      0.32      0.32       188
weighted avg       0.37      0.36      0.36       188
```

Nel Classification Report, vengono inserite diverse metriche. In particolare si trovano:

- Accuracy (o accuratezza): indica l'accuratezza del modello. Pertanto, la migliore accuratezza è 1, mentre la peggiore è 0. E' calcolata dalla seguente formula:

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

- Precision (o precisione): è l'abilità di un classificatore di non etichettare un'istanza positiva che è in realtà negativa. Per ogni classe è definito come il rapporto tra veri positivi e la somma di veri e falsi positivi. La formula di calcolo della precisione è la seguente:

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

- Recall (o richiamo): detta anche sensitivity o true positive, è la capacità di un classificatore di trovare tutte le istanze positive. Per ogni classe è definito come il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi negativi. Detto in altri termini, "per tutte le istanze che erano effettivamente positive, quale percentuale è stata classificata correttamente?". La formula di calcolo del richiamo è la seguente:

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

- F-score (o punteggio f): è una media armonica ponderata delle metriche Precision e Recall in modo tale che il punteggio migliore sia 1 e il peggiore sia 0. Come regola generale, la media ponderata di F1 dovrebbe essere utilizzata per confrontare i modelli di classificatore, non la precisione globale.

$$F - score = \frac{2\ x\ Recall\ x\ Precision}{Recall + Precision}$$

Complessivamente, i valori più alti di precision, recall, f1-score e support vengono ottenuti dal genere Action.

5.4 Una seconda versione del modello

Alla luce dei risultati illustrati nel sottocapitolo precedente, si è cercato di semplificare il modello. Nello specifico, appurato che aggiungere il Dropout e i diversi Regularizer non portava ad un miglioramento dei risultati, si è deciso di ridurre solamente il numero di

livelli della rete LSTM a parità delle altre condizioni eccetto due. Infatti, in primo luogo sono state aumentate a 70 le epoche in cui lanciare il modello in quanto esso risultava convergere a risultati migliori solo nell'eventualità di maggiori epoche a disposizione. In secondo luogo, è stato tolto il kernel_initializer="glorot_uniform" sui livelli di LSTM in quanto non portava ad un miglioramento.

```
model = Sequential();
model.add(Embedding(9, 32, input_length=X.shape[1]))
model.add(LSTM(50))
model.add(Dense(4, activation='softmax'));
print(model.summary());
model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 144, 32)           288

 lstm (LSTM)                 (None, 50)                16600

 dense (Dense)               (None, 4)                 204

=================================================================
Total params: 17,092
Trainable params: 17,092
Non-trainable params: 0
_________________________________________________________________
```

In questo caso il modello, avendo diminuito i livelli della rete LSTM sono diminuiti.

```
history = model.fit(X_train, Y_train, epochs=70, batch_size=5, validation_split=0.1,
callbacks=[EarlyStopping(monitor='val_loss', patience=300, min_delta=0.0001)])
```

```
Epoch 1/70
135/135 [==============================] - 9s 50ms/step - loss:
0.5592 - accuracy: 0.3096 - val_loss: 0.5254 - val_accuracy: 0.4400
Epoch 2/70
135/135 [==============================] - 6s 45ms/step - loss:
0.5361 - accuracy: 0.3437 - val_loss: 0.5171 - val_accuracy: 0.4667
Epoch 3/70
```

```
135/135 [==============================] - 6s 47ms/step - loss:
0.5249 - accuracy: 0.3956 - val_loss: 0.4962 - val_accuracy: 0.4933
Epoch 4/70
135/135 [==============================] - 6s 46ms/step - loss:
0.5203 - accuracy: 0.3881 - val_loss: 0.4917 - val_accuracy: 0.4667
Epoch 5/70
135/135 [==============================] - 6s 45ms/step - loss:
0.5192 - accuracy: 0.4015 - val_loss: 0.4895 - val_accuracy: 0.4933
Epoch 6/70
135/135 [==============================] - 6s 46ms/step - loss:
0.5108 - accuracy: 0.4119 - val_loss: 0.4924 - val_accuracy: 0.4667
Epoch 7/70
135/135 [==============================] - 6s 45ms/step - loss:
0.5138 - accuracy: 0.4163 - val_loss: 0.4870 - val_accuracy: 0.5067
Epoch 8/70
135/135 [==============================] - 6s 46ms/step - loss:
0.5069 - accuracy: 0.4252 - val_loss: 0.4926 - val_accuracy: 0.4933
Epoch 9/70
135/135 [==============================] - 6s 45ms/step - loss:
0.5088 - accuracy: 0.4444 - val_loss: 0.4917 - val_accuracy: 0.4400
Epoch 10/70
135/135 [==============================] - 6s 46ms/step - loss:
0.5077 - accuracy: 0.4415 - val_loss: 0.4933 - val_accuracy: 0.4933
Epoch 27/70
135/135 [==============================] - 6s 46ms/step - loss:
0.4622 - accuracy: 0.5037 - val_loss: 0.5155 - val_accuracy: 0.4667
Epoch 28/70
135/135 [==============================] - 6s 46ms/step - loss:
0.4595 - accuracy: 0.5274 - val_loss: 0.5137 - val_accuracy: 0.4933
Epoch 29/70
135/135 [==============================] - 6s 46ms/step - loss:
0.4531 - accuracy: 0.5230 - val_loss: 0.5232 - val_accuracy: 0.4400
Epoch 30/70
135/135 [==============================] - 6s 47ms/step - loss:
0.4478 - accuracy: 0.5274 - val_loss: 0.5296 - val_accuracy: 0.4667
Epoch 31/70
135/135 [==============================] - 7s 49ms/step - loss:
0.4455 - accuracy: 0.5422 - val_loss: 0.5110 - val_accuracy: 0.5067
Epoch 32/70
135/135 [==============================] - 6s 48ms/step - loss:
0.4349 - accuracy: 0.5674 - val_loss: 0.5265 - val_accuracy: 0.4000
Epoch 33/70
135/135 [==============================] - 6s 47ms/step - loss:
0.4334 - accuracy: 0.5585 - val_loss: 0.5312 - val_accuracy: 0.4533
Epoch 34/70
135/135 [==============================] - 6s 46ms/step - loss:
0.4242 - accuracy: 0.5689 - val_loss: 0.5394 - val_accuracy: 0.4400
Epoch 35/70
135/135 [==============================] - 6s 47ms/step - loss:
0.4122 - accuracy: 0.6015 - val_loss: 0.5324 - val_accuracy: 0.4533
Epoch 36/70
135/135 [==============================] - 6s 47ms/step - loss:
0.4019 - accuracy: 0.6089 - val_loss: 0.5350 - val_accuracy: 0.5067
Epoch 37/70
135/135 [==============================] - 6s 47ms/step - loss:
0.3980 - accuracy: 0.6237 - val_loss: 0.5415 - val_accuracy: 0.4267
```

Le epoche vengono troncate a 37 a causa dell'early stopping inserito. Come si può osservare, l'accuratezza di train passa da un valore iniziale del 30% per arrivare al 60%. L'accuratezza del validation ha invece valori compresi tra il 44% ed il 51%.

```
6/6 [==============================] - 1s 21ms/step - loss: 0.5727
- accuracy: 0.4202
Test set
  Loss: 0.573
  Accuracy: 0.420
```

Valutando il modello, si nota un miglioramento del valore di accuratezza complessiva di test che passa da un valore (del modello precedente) del 36% al 42%.

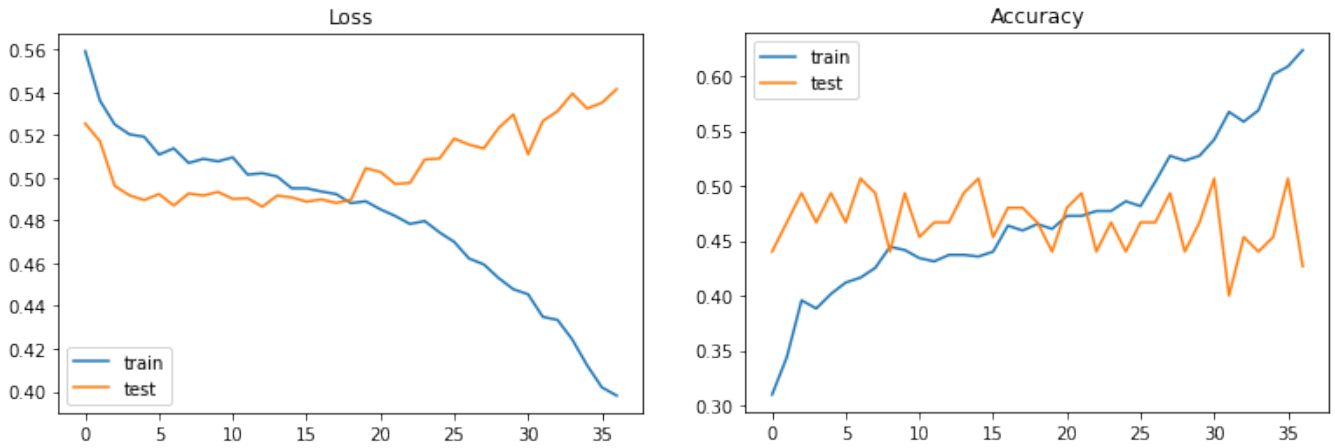

Figura 16 - Loss e Accuracy con LSTM=50

Osservando il grafico riportato nella *figura 18*, si nota che in questo caso l'overfitting, almeno in parte è stato limitato anche se non del tutto escluso. In questo caso però risulta l'inferiore l'accuratezza di train rispetto al primo caso in esame.

```
**** Confusion Matrix ****
[[29 22  7  3]
 [ 9 30 18  2]
 [17 21 19  0]
 [ 5  4  1  1]]
```

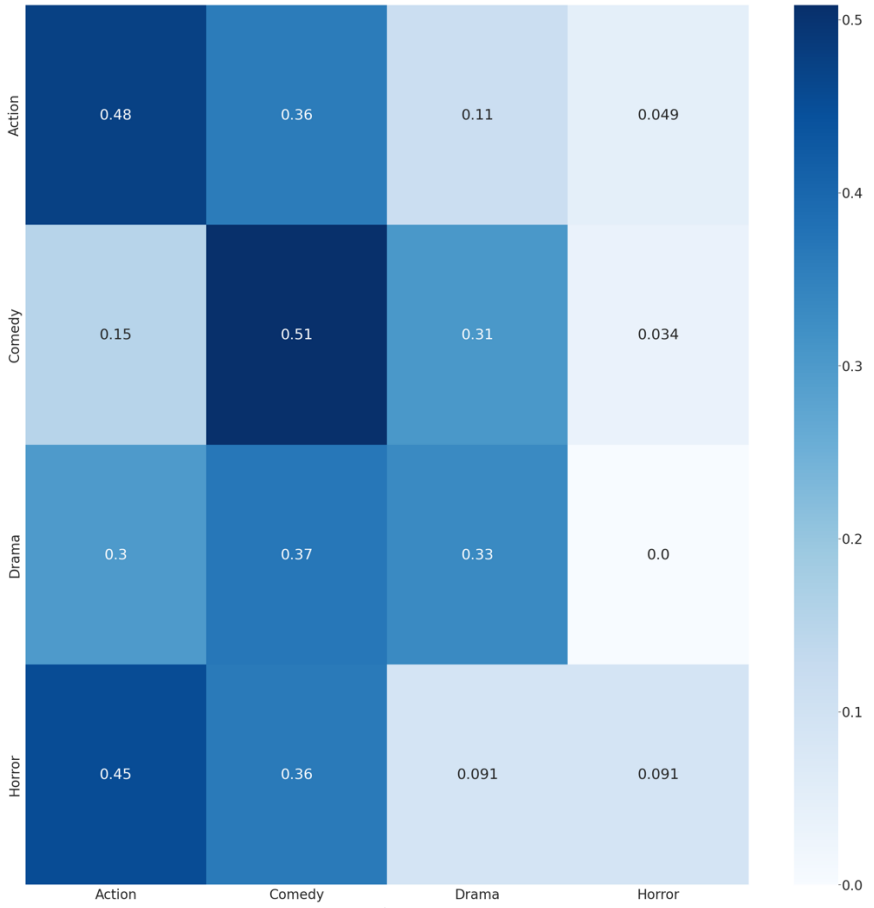

Figura 17 - Confusion Matrix con LSTM=50

Rispetto al caso precedente, nella *figura 19* si può osservare che i risultati sono migliorati in quanto i TP per quanto riguarda l'Action e i TP della classe Comedy raggiungono rispettivamente il 48% e il 51%. Il genere Drama invece continua ad essere confuso per il Comedy mentre quelli di genere Horror non è in grado di classificarli.

```
**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.48      0.48      0.48        61
      Comedy       0.39      0.51      0.44        59
       Drama       0.42      0.33      0.37        57
      Horror       0.17      0.09      0.12        11

    accuracy                           0.42       188
   macro avg       0.37      0.35      0.35       188
weighted avg       0.42      0.42      0.41       188
```

Analogamente a prima, ad eccezione del valore di recall, i risultati più soddisfacenti vengono ottenuti sul genere Action.

5.5 Cross Validation

La Cross Validation è un metodo statistico utilizzato per stimare l'abilità dei modelli di apprendimento automatico. Il principio su cui si basa è quello di non utilizzare l'intero set di dati durante l'addestramento di un modello ma una parte di essi viene rimossa prima dell'inizio dell'allenamento. Al termine della formazione, i dati precedentemente rimossi possono essere utilizzati per testare le prestazioni del modello appreso su nuovi dati.

In altre parole, la convalida del modello viene definita dal processo in cui un modello addestrato viene valutato con un set di dati di test. Il set di dati di test è una parte separata dello stesso set di dati da cui deriva il set di training. Lo scopo principale dell'utilizzo del set di dati di test è testare la capacità di generalizzazione di un modello addestrato.

In definitiva, il modello viene valutato solamente dopo che è stato addestrato. Insieme alla formazione del modello, la cross validation mira a trovare un modello ottimale con le migliori prestazioni.

Esistono diverse tipologie di convalida incrociata. Tra esse troviamo:

- K-fold Cross Validation: la convalida più diffusa che permette di definire k suddivisioni del dataset.
- Stratified Cross Validation: in cui in ogni piega o suddivisione la distribuzione dei campioni tra le classi viene mantenuta costante.
- LOOCV: in cui il numero delle suddivisioni è pari al numero delle osservazioni che abbiamo nel dataset.

Di seguito vengono riportate tre tra le prove effettuate con queste metodologie di Cross Validation. Si noti come i livelli di rete LSTM che si è scelto di utilizzare siano pari a 50 in quanto nei test precedenti dimostrava avere un minor overfitting e nel complesso performance migliori.

5.5.1 K-fold Cross Validation

La convalida incrociata k-fold prevede di dividere casualmente il set di dati in k gruppi, dette anche pieghe (fold) di dimensioni approssimativamente uguali. Il primo fold viene conservato per i test, mentre il modello viene addestrato sulle pieghe k-1.

Il processo viene ripetuto K volte e ogni volta per la convalida vengono utilizzate pieghe diverse o un diverso gruppo di punti dati. Ciò significa che ogni data point deve trovarsi in un set di test esattamente una volta e deve trovarsi in un set di allenamento k-1 volte. [37]

Nello specifico il modello analizzato è stato così impostato:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Bidirectional

from sklearn.model_selection import train_test_split

from keras.utils.np_utils import to_categorical

from keras.callbacks import EarlyStopping

from sklearn.model_selection import KFold

#Load .csv

df=pd.read_csv('//content/Dataset_completo.csv',squeeze=True, sep=";")

df.head()

# The maximum number of words to be used
MAX_NB_WORDS = 9
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 144
# This is fixed.
EMBEDDING_DIM = 32
#Spit the sequences
```

```
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters=' ', lower=True)
tokenizer.fit_on_texts(df['Sequence'])
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
#Definition of X
X = tokenizer.texts_to_sequences(df.Sequence.values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
#Definition of Y
Y = pd.get_dummies(df.Genere_1.values)
 print('Shape of label tensor:', Y.shape)
#Split X_train e X_test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,random_state = 0)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

Questa prima parte è analoga a quella già utilizzata nei due modelli precedenti.

```
# Define per-fold score containers
acc_per_fold = []
loss_per_fold = []
# Merge inputs and targets
inputs = np.concatenate((X_train, X_test), axis=0)
targets = np.concatenate((Y_train, Y_test), axis=0)
kf = KFold(n_splits=10)
kf.get_n_splits(X)
print(kf)
```

```
# Parse numbers as floats

input_train = X_train.astype('float32')

input_test = X_test.astype('float32')
```

In questa seconda parte viene definite il Kfold, con un numero di split pari a 10.

```
# K-fold Cross Validation model

fold_no = 1

KFold(n_splits=10, random_state=None, shuffle=False)

for train_index, test_index in kf.split(X):

        print("TRAIN:", train_index, "TEST:", test_index)

       X_train, X_test = X[train_index], X[test_index]

       Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

        model = Sequential()

        model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM,
 input_length= X.shape[1]))

        model.add(LSTM(50))

        model.add(Dense(4, activation="softmax"))

        model.compile(loss="binary_crossentropy",                optimizer="adam",
        metrics=["accuracy"])

        print(model.summary())

        # Generate a print

        print('------------------------------------------------------------------------')

         print(f'Training for fold {fold_no} ...')

        # Fit data to model
```

```
history = model.fit(X_train, Y_train, epochs=no_epochs, batch_size=batch_size,validation_split=0.1, callbacks=[EarlyStopping(monitor='val_loss', patience=40, min_delta=0.0001)])

# Generate generalization metrics

scores = model.evaluate(X_test, Y_test, verbose=0)

print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')

acc_per_fold.append(scores[1] * 100)

loss_per_fold.append(scores[0])
```

Output:

```
Epoch 34/70
152/152 [==============================] - 11s 71ms/step - loss: 0.4574
- accuracy: 0.5138 - val_loss: 0.5960 - val_accuracy: 0.3176
Epoch 35/70
152/152 [==============================] - 11s 71ms/step - loss: 0.4341
- accuracy: 0.5507 - val_loss: 0.5988 - val_accuracy: 0.3176
Epoch 36/70
152/152 [==============================] - 11s 71ms/step - loss: 0.4352
- accuracy: 0.5415 - val_loss: 0.6030 - val_accuracy: 0.2941
Epoch 37/70
152/152 [==============================] - 11s 71ms/step - loss: 0.4250
- accuracy: 0.5639 - val_loss: 0.6116 - val_accuracy: 0.3412
Epoch 38/70
152/152 [==============================] - 11s 72ms/step - loss: 0.4140
- accuracy: 0.5929 - val_loss: 0.6177 - val_accuracy: 0.3647
Epoch 39/70
152/152 [==============================] - 11s 72ms/step - loss: 0.4075
- accuracy: 0.5955 - val_loss: 0.6310 - val_accuracy: 0.3294
Epoch 40/70
152/152 [==============================] - 11s 71ms/step - loss: 0.4019
- accuracy: 0.6047 - val_loss: 0.6280 - val_accuracy: 0.4000
Epoch 41/70
152/152 [==============================] - 11s 71ms/step - loss: 0.3940
- accuracy: 0.6258 - val_loss: 0.6327 - val_accuracy: 0.3529
Epoch 42/70
152/152 [==============================] - 11s 72ms/step - loss: 0.3893
- accuracy: 0.6298 - val_loss: 0.6376 - val_accuracy: 0.3765
Epoch 43/70
152/152 [==============================] - 11s 71ms/step - loss: 0.3792
- accuracy: 0.6377 - val_loss: 0.6629 - val_accuracy: 0.3529
Epoch 44/70
152/152 [==============================] - 11s 71ms/step - loss: 0.3754
- accuracy: 0.6574 - val_loss: 0.6796 - val_accuracy: 0.3294
Score for fold 8: loss of 0.5808522701263428; accuracy of
47.8723406791687%
3/3 [==============================] - 0s 24ms/step - loss: 0.5809 -
accuracy: 0.4787
```

L'output riportato nella *figura 20* rappresenta il caso migliore ottenuto inserendo il KFold. Come si può osservare l'accuratezza del test arriva ad un valore medio del 48%. Il risultato non è sorprendente ma considerati tutti i limiti del dataset e dell'analisi che si vedranno successivamente è comunque un buon miglioramento.

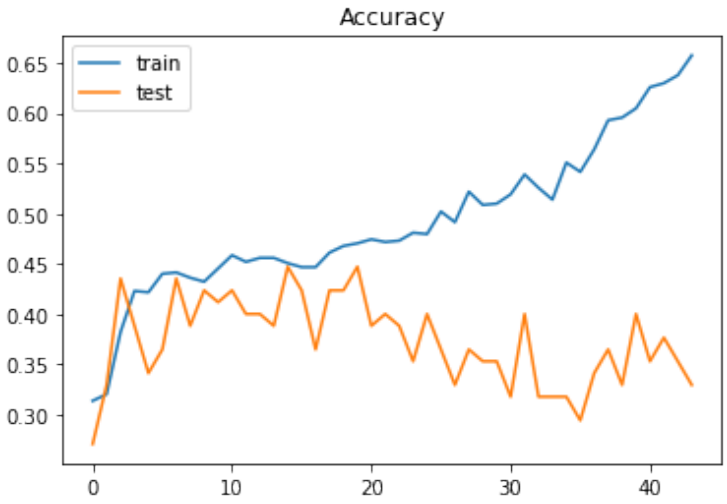

Figura 18 - Accuracy KFold

Inoltre, guardando i valori di recall, si può osservare che su due generi su quattro il valore di recall supera il 50%. Il valore di recall, come già detto, indica la percentuale degli elementi "positivi" realmente identificati come "positivi". Questo indica ad esempio per il genere Drama che il 63% delle clip identificate con quel genere, effettivamente appartiene all'etichetta precedentemente data.

Il valore dell'accuratezza di test complessivo è ovviamente influenzato dalla totale incapacità di identificare il genere Horror. Esso essendo una parte molto piccola del dataset non ha sufficienti dati a disposizione per far si che il modello impari a riconoscerlo.

Su questo aspetto, nei capitoli seguenti verranno proposte altre due alternative di Cross Validation in cui sarà mostrato come, i risultati possono essere migliorati.

Invece, per quanto riguarda i valori di precision, anche qui due generi superano il 50%. Questo significa che il modello, nel 52% dei casi, ad esempio non identifica una clip come Action se effettivamente questa non lo è.

```
**** Confusion Matrix ****
[[11  6  8  1]
 [ 4 15 11  0]
 [ 3  7 19  1]
 [ 3  5  0  0]]


**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.52      0.42      0.47        26
      Comedy       0.45      0.50      0.48        30
       Drama       0.50      0.63      0.56        30
      Horror       0.00      0.00      0.00         8

    accuracy                           0.48        94
   macro avg       0.37      0.39      0.38        94
weighted avg       0.45      0.48      0.46        94
```

Inoltre, testando questo modello con un numero inferiore di livelli di reti LSTM, ponendolo pari a 30, risulta avere un'accuracy di train più bassa ma complessivamente l'accuratezza di test supera il 50%:

```
Epoch 36/70
127/127 [==============================] - 9s 61ms/step - loss: 0.4815
- accuracy: 0.4993 - val_loss: 0.5455 - val_accuracy: 0.4000
Epoch 37/70
127/127 [==============================] - 9s 62ms/step - loss: 0.4814
- accuracy: 0.4888 - val_loss: 0.5478 - val_accuracy: 0.4353
Epoch 38/70
127/127 [==============================] - 9s 62ms/step - loss: 0.4816
- accuracy: 0.4954 - val_loss: 0.5242 - val_accuracy: 0.4118
Epoch 39/70
127/127 [==============================] - 9s 61ms/step - loss: 0.4809
- accuracy: 0.5046 - val_loss: 0.5449 - val_accuracy: 0.4118
Epoch 40/70
127/127 [==============================] - 9s 62ms/step - loss: 0.4805
- accuracy: 0.4875 - val_loss: 0.5464 - val_accuracy: 0.4235
Epoch 41/70
127/127 [==============================] - 9s 61ms/step - loss: 0.4801
- accuracy: 0.5046 - val_loss: 0.5452 - val_accuracy: 0.4118
Epoch 42/70
127/127 [==============================] - 9s 62ms/step - loss: 0.4800
- accuracy: 0.4954 - val_loss: 0.5555 - val_accuracy: 0.4000
Epoch 43/70
127/127 [==============================] - 9s 61ms/step - loss: 0.4797
- accuracy: 0.5145 - val_loss: 0.5469 - val_accuracy: 0.4118
Score for fold 1: loss of 0.50653487443923295; accuracy of
51.06382966041565%
3/3 [==============================] - 0s 11ms/step - loss: 0.5065 -
accuracy: 0.5106
```

```
**** Confusion Matrix ****
[[10  7  9  0]
 [ 3 17  8  0]
 [ 2 10 21  0]
 [ 3  0  4  0]]
```

```
**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.56      0.38      0.45        26
      Comedy       0.50      0.61      0.55        28
       Drama       0.50      0.64      0.56        33
      Horror       0.00      0.00      0.00         7

    accuracy                           0.51        94
   macro avg       0.39      0.41      0.39        94
weighted avg       0.48      0.51      0.49        94
```

In questo caso si può osservare come i valori di Precision siano superiori o uguali al 50% in tre generi su quattro. Questo è ciò che ci si aspettava in quanto l'unico genere in cui non riesce a raggiungere la soglia del 50% è l'Horror che nel dataset costruito ha davvero pochi record.

Inoltre, il valore di f1-score, che come già detto è una media tra i valori di precisione e di recall, è superiore al 50% per il genere Comedy e Drama.

I valori di Recall per Drama e Comedy sono soddisfacenti e dimostrano che il modello costruito in questo modo è in grado, in più della maggioranza dei casi, di identificare quella determinata sequenza con il genere.

5.5.2 Stratified Cross Validation

La stratificazione è una tecnica in cui si riordinano i dati in modo tale che ogni fold abbia una buona rappresentazione dell'intero set di dati: ogni fold deve avere almeno m istanze di ogni classe. Questo approccio garantisce che una classe di dati non sia sovrapesata soprattutto quando la variabile target è sbilanciata. Inoltre, aiuta a ridurre sia il bias che la varianza. [37]

Nel caso in esame, il modello è stato realizzato analogamente al KFold inserendo questa parte di codice:

```
skf = StratifiedKFold(n_splits=10)
```

```
skf.get_n_splits(inputs, targets)
```

Output:

```
Epoch 60/70
152/152 [==============================] - 11s 71ms/step - loss: 0.2437
- accuracy: 0.8142 - val_loss: 0.8236 - val_accuracy: 0.3294
Epoch 61/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2433
- accuracy: 0.8155 - val_loss: 0.7882 - val_accuracy: 0.3176
Epoch 62/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2703
- accuracy: 0.7773 - val_loss: 0.7905 - val_accuracy: 0.3294
Epoch 63/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2707
- accuracy: 0.7839 - val_loss: 0.8054 - val_accuracy: 0.3176
Epoch 64/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2293
- accuracy: 0.8366 - val_loss: 0.8440 - val_accuracy: 0.3059
Epoch 65/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2162
- accuracy: 0.8472 - val_loss: 0.8558 - val_accuracy: 0.3176
Epoch 66/70
152/152 [==============================] - 11s 70ms/step - loss: 0.2069
- accuracy: 0.8590 - val_loss: 0.8461 - val_accuracy: 0.3765
Epoch 67/70
152/152 [==============================] - 11s 71ms/step - loss: 0.1952
- accuracy: 0.8735 - val_loss: 0.8937 - val_accuracy: 0.3059
Epoch 68/70
152/152 [==============================] - 11s 71ms/step - loss: 0.1907
- accuracy: 0.8762 - val_loss: 0.9027 - val_accuracy: 0.3059
Epoch 69/70
152/152 [==============================] - 11s 71ms/step - loss: 0.1869
- accuracy: 0.8748 - val_loss: 0.9222 - val_accuracy: 0.3176
Epoch 70/70
152/152 [==============================] - 11s 70ms/step - loss: 0.1854
- accuracy: 0.8748 - val_loss: 0.8860 - val_accuracy: 0.3176
Score for fold 8: loss of 0.7253977060317993; accuracy of
45.7446813583374%
3/3 [==============================] - 0s 22ms/step - loss: 0.7254 -
accuracy: 0.4574
```

L'output riportato nella *figura 21* presenta il miglior risultato ottenuto tramite lo Stratified KFold. L'accuratezza complessiva del modello è del 46% e i valori di accuracy di train arrivano all'89%. I dati di validation invece si attestano sempre in un intervallo compreso tra il 30% e il 45%.

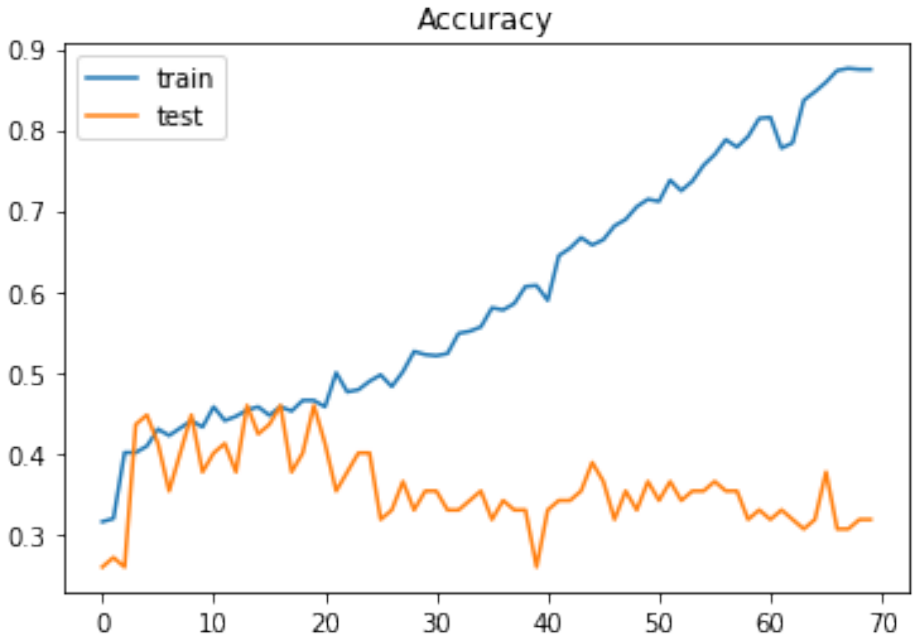

Figura 19 - Accuracy SKF

```
**** Confusion Matrix ****
[[13  7  6  0]
 [ 6 17  7  0]
 [ 8  9 12  1]
 [ 3  3  1  1]]
**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.43      0.50      0.46        26
      Comedy       0.47      0.57      0.52        30
       Drama       0.46      0.40      0.43        30
      Horror       0.50      0.12      0.20         8

    accuracy                           0.46        94
   macro avg       0.47      0.40      0.40        94
weighted avg       0.46      0.46      0.45        94
```

In questo caso invece, i valori sono leggermente più bassi del caso precedente. Se prima il Drama otteneva dei buoni valori di recall, in questo caso sono Comedy e Action ad ottenere quelli più alti. E' interessante notare il dato di Precision su Horror: la precision, appunto, è l'abilità di un classificatore di non etichettare un'istanza positiva che è in realtà negativa ed in questo caso raggiunge il 50%. Infatti, questo miglioramento è riconducibile alla natura stessa dello Stratified KFold: imponendo che in ogni classe debbano esserci almeno m istanze di ogni valore possibile di Y, il modello riesce, anche se in piccola parte, a riconoscere una clip di genere Horror.

Inoltre, merita particolare attenzione anche il dato della precision sul genere Horror, pari addirittura al 50%: infatti, se una clip non è Horror, nel 50% dei casi non verrà etichettata come Horror.

5.5.3 LOOCV

Nella convalida LOOCV dividiamo il set di dati in due parti. In una parte abbiamo un'unica osservazione, che sono i nostri dati di test e nell'altra parte, abbiamo tutte le altre osservazioni del set di dati di allenamento (set di training). Se si dispone di un set di dati con n osservazioni, i dati di allenamento contengono n-1 osservazioni e i dati di test contengono un'unica osservazione. [37]

Per realizzare questo tipo di convalida, il modello è rimasto inalterato ai due casi precedenti ma è stato implementato il LOOCV tramite questo codice:

```
kf = KFold(n_splits=10)

kf.get_n_splits(X)
```

```
Epoch 60/70
127/127 [==============================] - 9s 70ms/step - loss: 0.6400
- accuracy: 0.7453 - val_loss: 1.8634 - val_accuracy: 0.4085
Epoch 61/70
127/127 [==============================] - 9s 69ms/step - loss: 0.5462
- accuracy: 0.7975 - val_loss: 1.9049 - val_accuracy: 0.4085
Epoch 62/70
127/127 [==============================] - 9s 68ms/step - loss: 0.5109
- accuracy: 0.8149 - val_loss: 1.9391 - val_accuracy: 0.4366
Epoch 63/70
127/127 [==============================] - 9s 71ms/step - loss: 0.4927
- accuracy: 0.8133 - val_loss: 2.0272 - val_accuracy: 0.3944
Epoch 64/70
127/127 [==============================] - 9s 70ms/step - loss: 0.4729
- accuracy: 0.8370 - val_loss: 2.0126 - val_accuracy: 0.4085
Epoch 65/70
127/127 [==============================] - 9s 68ms/step - loss: 0.4593
- accuracy: 0.8402 - val_loss: 2.0237 - val_accuracy: 0.4507
Epoch 66/70
127/127 [==============================] - 9s 70ms/step - loss: 0.5284
- accuracy: 0.7943 - val_loss: 2.1032 - val_accuracy: 0.3803
Epoch 67/70
127/127 [==============================] - 9s 70ms/step - loss: 0.4395
- accuracy: 0.8528 - val_loss: 2.1957 - val_accuracy: 0.4225
Epoch 68/70
127/127 [==============================] - 9s 70ms/step - loss: 0.4049
- accuracy: 0.8671 - val_loss: 2.1101 - val_accuracy: 0.4085
Epoch 69/70
```

```
127/127 [==============================] - 9s 70ms/step - loss: 0.4009
- accuracy: 0.8623 - val_loss: 2.2847 - val_accuracy: 0.3803
Epoch 70/70
127/127 [==============================] - 9s 69ms/step - loss: 0.4048
- accuracy: 0.8639 - val_loss: 2.4245 - val_accuracy: 0.3099
Score for fold 1: loss of 1.6447070837020874; accuracy of
45.95744609832764%
8/8 [==============================] - 0s 20ms/step - loss: 1.6447 -
accuracy: 0.4796
```

Il valore complessivo sulla parte di Test è del 48%. Complessivamente dunque il modello, testato su quei determinati dati di test, nel 48% dei casi riesce ad etichettare correttamente il genere. L'accuracy della parte di Train rimane sempre alta, come si può vedere nella *figura 22* ed in continua crescita all'aumentare delle epoche, è alla fine, infatti, pari all'86%.

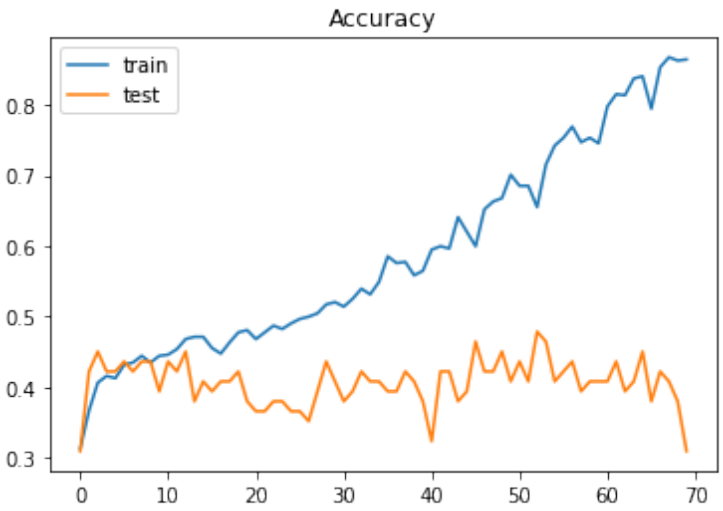

Figura 20 - Accuracy LOOCV

```
**** Confusion Matrix ****
[[35 14 18  8]
 [11 36 18  1]
 [14 25 34  5]
 [ 5  3  5  3]]

**** Classification Report ****
              precision    recall  f1-score   support

      Action       0.54      0.47      0.50        75
      Comedy       0.46      0.55      0.50        66
       Drama       0.45      0.44      0.44        78
      Horror       0.18      0.19      0.18        16

    accuracy                           0.46       235
   macro avg       0.41      0.41      0.41       235
weighted avg       0.47      0.47      0.47       235
```

In questo caso, osservando l'indicatore f1-score che è una media armonica ponderata delle metriche Precision e Recall è pari al 50% in due generi su quattro. In questo ultimo caso, anche l'Horror inizia a predire correttamente qualcosa, pur nelle sue limitazioni dovute al basso numero di clip etichettati con quel genere.

I risultati ottenuti grazie a questa tipologia di Cross Validation riportano delle analogie con i risultati ottenuti nello SKF: anche in questo caso, seppur in minima parte, i valori delle clip Horror iniziano ad essere predette correttamente nel 19% dei casi.

5.6 Conclusioni dell'analisi e limiti

Alla luce di tutti i risultati ottenuti, è importante evidenziare i limiti di quest'analisi. Innanzitutto, i valori ottenuti di accuracy vanno visionati in relazione al Dataset su cui le analisi sono state svolte.

Il Dataset è stato creato manualmente prendendo le clip direttamente da Youtube e la loro ricerca è stata piuttosto lunga poiché il canale su cui sono state prese mano a mano che si scendeva nella ricerca dei video, non era in grado di caricare correttamente la pagina a causa dell'elevato sforzo richiesto ai pc.

Inoltre, uno dei limiti più importanti è dovuto sicuramente alla ristrettezza del Dataset. 937 record probabilmente non sono sufficienti per avere risultati sorprendenti, in relazione anche al fatto che di ogni record si è riuscito ad individuare poche informazioni rilevanti volte alla sua caratterizzazione.

I valori di accuracy del Validation test si attestano sempre in un range tra il 35% ed il 45%. Anche questo è attribuibile al fatto che il dataset abbia troppi pochi dati a disposizione e per questo risulta difficile per il modello predire correttamente il genere.

Un altro limite è rappresentato dalla scarsità delle clip identificate come Horror che dunque pregiudicano la maggior parte delle analisi. Come si è visto, in tutti i tre metodi utilizzati di Cross Validation, i risultati mostravano valori sufficienti per almeno due generi su quattro.

A seguire, il fatto che le sequenze avessero tutte lunghezze differenti è un limite per il tipo di analisi da svolgere. Infatti, l'algoritmo sopra implementato, per riuscire a predire le etichette necessita di una lunghezza fissa della sequenza. Ovviamente, il codice è stato

testato su lunghezza di sequenze differenti, a partire da 10 per arrivare a 144 che è il valore della lunghezza della sequenza di inquadrature più lunga presente nel dataset. Le performance migliori continuavano ad essere quelle ottenute con la lunghezza di 144 in quanto in corrispondenza delle inquadrature mancanti, ad esempio di una stringa di lunghezza 142 venivano inseriti dei valori nulli.

Oltre ad essere sbilanciato dal punto di vista dei generi, il Dataset risulta esserlo anche dal punto di vista della numerosità totale delle inquadrature di diverso tipo. La prevalenza di inquadrature di tipo 5 o 6, che sono preponderanti rispetto alle altre, ovviamente influenza le performance del modello.

In conclusione, il limite principale è rappresentato dal fatto che molte delle clip presenti nel Dataset sono dialoghi e per questo contengono inquadrature di tipo 5 o 6, che il modello non riesce a distinguere per genere in quanto potrebbero appartenere ad un Action come ad un Comedy o un Drama.

Le analisi, in conclusione, vanno viste in relazione a tutti i limiti sopra elencati e per questo, pur non raggiungendo dei valori di accuracy molto elevati sono da considerarsi sufficienti poiché, probabilmente, se applicate a dei Dataset più numerosi o più vari potrebbero dare risultati ancora migliori.

Capitolo 6: Classificazione delle sequenze per la previsione della prossima inquadratura

6.1 Pre-processing

In relazione a tutte le considerazioni fatte precedentemente, si è provato a semplificare le analisi. Ad esempio, ci si è chiesti se dando in pasto al modello un vettore X di una sequenza di n elementi, esso fosse in grado di predire quello successivo sia fornendo in input anche l'informazione del genere sia no e anche all'aumentare o al diminuire di n.

Per fare ciò la prima cosa che è stata fatta è stata quella di suddividere l'intero dataset in sequenze di n elementi Di seguito è illustrato il codice utilizzato:

```
import numpy as np
```

```
import pandas as pd

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences
```

Nella prima parte vengono importate tutte le librerie utili.

```
df=pd.read_csv('/Users/letiziademarchi/Desktop/Tesi Magistrale/Dataset/Dataset_completo811.csv',squeeze=True, sep=";")

sequences=[]

y=[]

def chunks(l, n):
    n_items = len(l)
    if n_items % n:
        n_pads = n - n_items % n
    else:
        n_pads = 0
    l = l + ['<PAD>' for _ in range(n_pads)]
    for i in range(0, len(l), n):
        yield l[i:i + n]
        sequences.append(l[i:i + n])

df['Sequence1'] = df['Sequence1'].str.split().apply(lambda x: list(chunks(x, 3)))

df = df.explode('Sequence1').reset_index(drop=True)

df['Sequence1'] = df['Sequence1'].apply(' '.join)

print(df)
```

Questo codice prende in pasto l'intera sequenza, ad esempio [5, 6, 2, 2, 6, 6, 6, 5, 5, 2, 4, 4] e la divide a gruppi di 5. Poiché le sequenze nel dataset non avevano una lunghezza

totale divisibile per 5, si è pensato di inserire, per i numeri mancanti dell'ultima sequenza <PAD> che indica elemento mancante. Nel caso in esempio la divisione sarà la seguente:

5, 6, 2, 2, 6	Action
6, 6, 5, 5, 2	Action
4, 4 <PAD> <PAD>	Action

```
df.to_csv('/Users/letiziademarchi/Desktop/Tesi
Magistrale/Dataset/Dataset_diviso3.csv', index=False, sep=";")
```

In seguito, il file è stato scritto in un file .csv in modo da poterlo elaborare su Excel. Qui è stato inserito in una colonna "UltimaSequenza" l'elemento successivo della sequenza da predire. Ad esempio, nel caso sopra riportato il valore di UltimaSequenza è stato posto uguale a 6. Inoltre, sono stati cancellati tutti i record contenenti <PAD> poiché inutili ai fine dell'analisi.

Volendo comprendere se l'inserimento del genere nella sequenza di input aiuti o meno la predizione della prossima inquadratura, i generi sono stati trasformati in valori:

- Action=10
- Comedy=11
- Drama=12
- Horror=13

Una volta trasformati i generi in valori, sono stati posti all'inizio della sequenza. Ovvero, se prima la sequenza considerata era 5, 6, 2, 2, 6 di genere Action, ora queste due informazioni sono contenute in una singola stringa [10, 5, 6, 2, 2, 6].

Questo trattamento dei dati è stato fatto per tre n differenti: n=5, n=3, n=10.

6.2 Il modello

Per l'implementazione di quest'analisi è stato realizzato un modello leggermente diverso da quello adottato per la prima analisi. Per cercare di migliorare i risultati, all'interno del codice ogni split viene testato su due modelli differenti: il primo contiene le semplici reti

LSTM e i livelli Embedding già utilizzati nel modello precedente, il secondo invece, aggiunge il meccanismo di attenzione.

In una semplice architettura Encoder-Decoder il decodificatore dovrebbe iniziare a fare previsioni guardando solo l'output finale del passo dell'encoder che ha informazioni condensate.

Al contrario, l'idea principale del meccanismo di attenzione è consentire al decodificatore di accedere selettivamente alle informazioni del codificatore durante la decodifica. Ciò si ottiene costruendo un vettore di contesto diverso per ogni passo temporale del decoder, calcolandolo in funzione dello stato nascosto precedente e di tutti gli stati nascosti dell'encoder, assegnando loro pesi addestrabili.

In questo modo, il meccanismo di attenzione assegna un'importanza diversa ai diversi elementi della sequenza di input e presta maggiore attenzione agli input più rilevanti.

Di seguito viene riportato il modello utilizzato. Le sequenze scelte sono quelle con n=5 in quanto dopo diverse prove sono risultate le più performanti, inoltre per aumentare la validità del modello è stato inserito il KFold:

```
import numpy

import numpy as np

from tensorflow.keras import Sequential

from tensorflow.keras.callbacks import Callback

from tensorflow.keras.datasets import imdb

from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM

from tensorflow.keras.preprocessing import sequence

import pandas as pd

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from sklearn.model_selection import train_test_split

from sklearn.model_selection import LeaveOneOut
```

```
import matplotlib.pyplot as plt

from sklearn.model_selection import KFold

from attention import Attention

df=pd.read_csv('/Users/letiziademarchi/Desktop/Tesi
Magistrale/Dataset/Dataset_completo812.csv',squeeze=True, sep=";")

df.head()
```

In questa prima parte vengono importate le librerie e il dataset da utilizzare.

```
X=df.Sequence1.values

y=df['UltimaSequenzaOK']

X = tokenizer.texts_to_sequences(df.Sequence1.values)

X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)

print('Shape of data tensor:', X.shape)

print(X)

Y = pd.get_dummies(df.UltimaSequenzaOK.values)

print('Shape of label tensor:', Y.shape)
```

In questa seconda parte i dati vengono convertiti in un formato consono per l'analisi.

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42)
```

Il dataset viene diviso tra la parte di Train e di Test.

```
# Define per-fold score containers

acc_per_fold = []

loss_per_fold = []
```

```
# Merge inputs and targets

inputs = np.concatenate((X_train, X_test), axis=0)

targets = np.concatenate((Y_train, Y_test), axis=0)

kf = KFold(n_splits=10)

kf.get_n_splits(X)

print(kf)

# Parse numbers as floats

input_train = X_train.astype('float32')

input_test = X_test.astype('float32')
```

In questa parte viene inizializzato il Kfold.

```
KFold(n_splits=10, random_state=None, shuffle=False)
for train_index, test_index in kf.split(X):
   X_train, X_test = X[train_index], X[test_index]
   Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]
   def train_and_evaluate_model (add_attention=True):
      numpy.random.seed(7)
      # load the dataset but only keep the top n words, zero the rest
      top_words = 9
      #(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=top_words)
      # truncate and pad input sequences
      max_review_length = 5
      #x_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
      #x_test = sequence.pad_sequences(x_test, maxlen=max_review_length)
      # create the model
      embedding_vector_length = 32
       model = Sequential([ Embedding(top_words, embedding_vector_length,
       input_length=max_review_length),
# attention vs no attention. same number of parameters so fair comparison.
```

```
        *([LSTM(100, return_sequences=True), Attention()] if add_attention
                    else [LSTM(100), Dense(8, activation='relu')]),
                            Dense(16, activation='relu')
                    Dense(8, activation='softmax')])
                     model.compile(loss='categorical_crossentropy', optimizer='adam',
                     metrics=['accuracy'])
                      print(model.summary())


    class RecordBestTestAccuracy(Callback):
        def __init__(self):
            super().__init__()
            self.val_accuracies = []
            self.val_losses = []
        def on_epoch_end(self, epoch, logs=None):
            self.val_accuracies.append(logs['val_accuracy'])
            self.val_losses.append(logs['val_loss'])
    rbta = RecordBestTestAccuracy()
                history=model.fit(X_train, Y_train, validation_split=0.1, epochs=50, batch_size=5,
         callbacks=[rbta])
    print(f"Max Test Accuracy: {100 * np.max(rbta.val_accuracies):.2f} %")
    print(f"Mean Test Accuracy: {100 * np.mean(rbta.val_accuracies):.2f} %")
    plt.title('Loss')
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.legend()
    plt.show();
    plt.title('Accuracy')
    plt.plot(history.history['accuracy'], label='train')
    plt.plot(history.history['val_accuracy'], label='test')
    plt.legend()
    plt.show();
def main():
```

```
    train_and_evaluate_model (add_attention=False)
    train_and_evaluate_model (add_attention=True)
if __name__ == '__main__':
    main()
```

Questo modello, come si evince dal codice, utilizza due approcci differenti in cui nel secondo, viene aggiunto il Meccanismo di Attenzione. Infatti, il codice fa riferimento ad un altro modello implementato in un file .py che permette, qualora sia attivato, di effettuare l'analisi inserendo il Meccanismo di Attenzione.

```
    from tensorflow.keras.layers import Dense, Lambda, Dot, Activation, Concatenate
    from tensorflow.keras.layers import Layer
    class Attention(Layer):
        def __init__(self, units=128, **kwargs):
            self.units = units
            super().__init__(**kwargs)
        def __call__(self, inputs):
            """
            Many-to-one attention mechanism for Keras.
            @param inputs: 3D tensor with shape (batch_size, time_steps, input_dim).
            @return: 2D tensor with shape (batch_size, 128)
            @author: felixhao28, philipperemy.
            """
            hidden_states = inputs
            hidden_size = int(hidden_states.shape[2])
            # Inside dense layer
            #           hidden_states          dot             W         =>        score_first_part
            # (batch_size, time_steps, hidden_size) dot (hidden_size, hidden_size) =>
    (batch_size, time_steps, hidden_size)
            # W is the trainable weight matrix of attention Luong's multiplicative style score
            score_first_part           =           Dense(hidden_size,           use_bias=False,
    name='attention_score_vec')(hidden_states)
            #         score_first_part        dot      last_hidden_state     => attention_weights
```

```
        # (batch_size, time_steps, hidden_size) dot (batch_size, hidden_size) => (batch_size, time_steps)
        h_t = Lambda(lambda x: x[:, -1, :], output_shape=(hidden_size,), name='last_hidden_state')(hidden_states)
        score = Dot(axes=[1, 2], name='attention_score')([h_t, score_first_part])
        attention_weights = Activation('softmax', name='attention_weight')(score)
        # (batch_size, time_steps, hidden_size) dot (batch_size, time_steps) => (batch_size, hidden_size)
        context_vector = Dot(axes=[1, 1], name='context_vector')([hidden_states, attention_weights])
        pre_activation = Concatenate(name='attention_output')([context_vector, h_t])
        attention_vector = Dense(self.units, use_bias=False, activation='tanh', name='attention_vector')(pre_activation)
        return attention_vector
    def get_config(self):
        return {'units': self.units}
    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

Questo modello è stato applicato sia nel caso della sequenza con n=5 senza l'informazione del genere che nel caso di n=5 ma con l'informazione sul genere.

6.3 Analisi e interpretazione dei risultati

Di seguito si riportano i due grafici dove sull'asse delle Y vengono riportati i valori medi di Accuracy raggiunti in ogni fold testato.

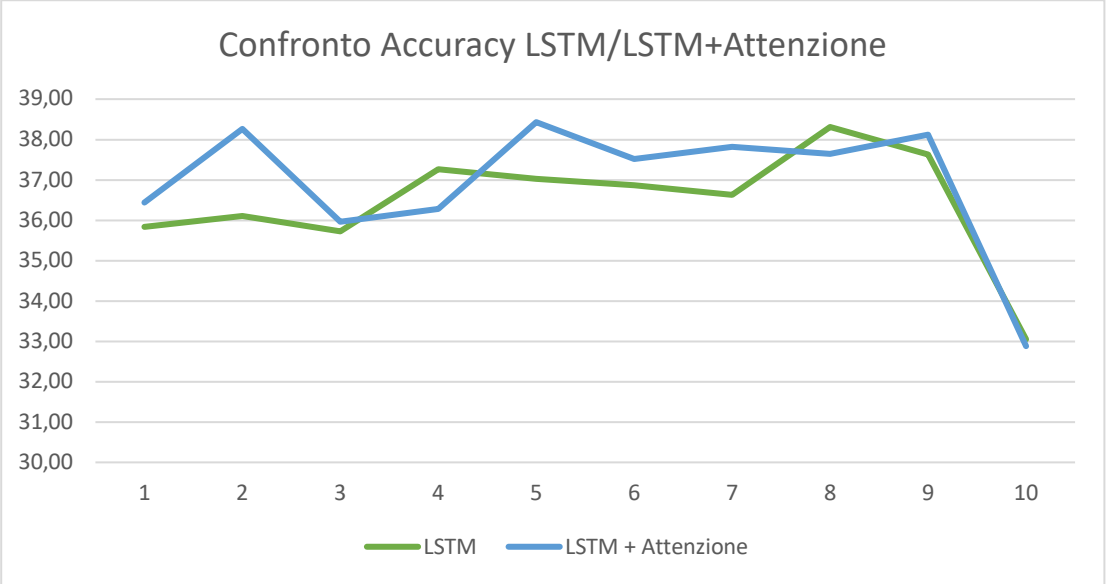

Figura 21 - Confronto accuracy senza il genere

Questo primo grafico (*figura 23*) mostra i risultati ottenuti testando il modello senza inserire l'informazione sul genere. I valori medi di accuracy raggiunti sono tra il 36% e il 38% ma non ci si aspettava risultati molto diversi viste le complessità già prima elencate. Quello che è interessante notare è che in quasi tutti i fold ad eccezione di 2, i valori di accuracy raggiunti inserendo il Meccanismo di Attenzione sono sempre maggiori.

Questo indica che il Meccanismo di Attenzione ha un impatto positivo sull'analisi e che in presenza di maggiori dati o di diversa natura potrebbe aiutare il modello a performare meglio.

A questo punto ci si è chiesti se inserendo l'informazione sul genere all'inizio della sequenza, essa aiutasse il modello a identificare meglio la seguente inquadratura. Di seguito viene mostrato il medesimo grafico di sopra ma nel caso appena descritto:

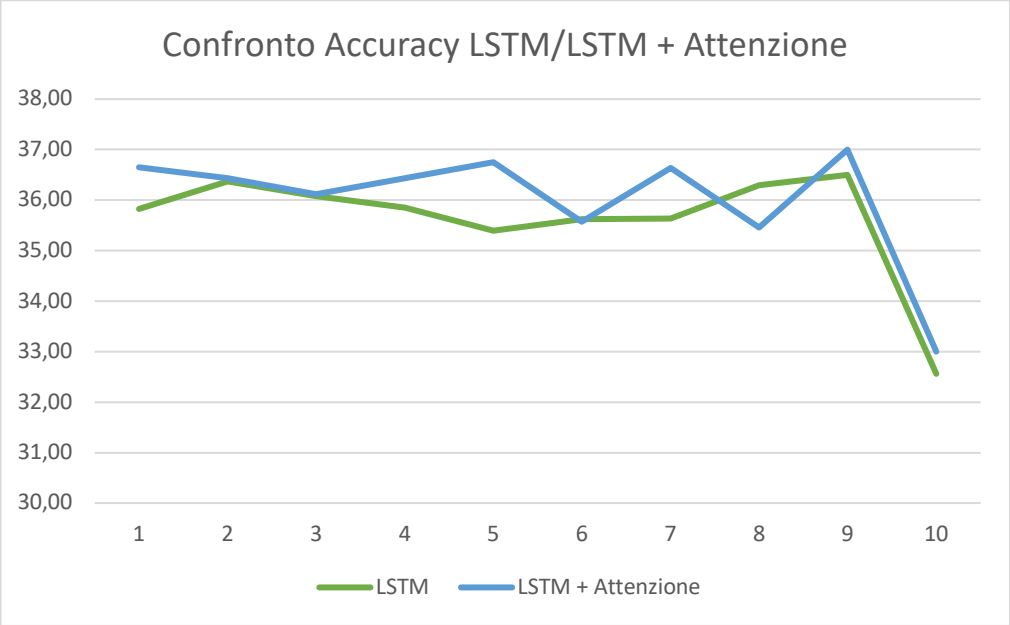

Figura 22 - Confronto Accuracy con il genere

Anche in questo grafico (*figura 24*), si può notare come se ai livelli LSTM viene aggiunto il meccanismo di Attenzione, esso sembra performare meglio. Infatti, ad eccezione di un fold, la linea azzurra che indica l'andamento dell'accuratezza una volta inserito il Meccanismo di Attenzione è sempre sopra la linea verde.

Confrontando invece i due tipi di analisi, complessivamente il modello che risulta avere un'accuratezza complessiva più alta è quello senza il genere. Infatti, anche alla luce dei risultati dell'analisi precedente, il genere non è così influente alla determinazione del tipo di inquadratura di una sequenza e questo è evidente in questa seconda analisi in cui il modello non risulta sensibile al genere.

Inoltre, si può dedurre che al modello, più informazioni vengono date più tende a confondersi: questo indica che dovendo tenere in considerazione anche il genere esso non riesce a classificare meglio del caso precedente la previsione della prossima inquadratura.

6.4 Conclusione dell'analisi e limiti

Questa seconda analisi effettuata aveva come scopo quello di verificare se, semplificato il modello, ovvero se diviso in sequenze di n parti uguali e se aiutato con l'inserimento del genere nella sequenza analizzare presentasse performance migliori di quello precedentemente illustrato.

Le conclusioni che sono emerse è che anche in questo caso emergono tutte le limitatezze del dataset, in particolare è importante notare come qui, escludendo il genere, siano davvero impattanti le diverse distribuzioni delle tipologie di inquadrature.

Infatti, splittando le sequenze in parti più piccole, il modello non riesce tendenzialmente a predire in modo corretto il tipo di inquadratura seguente. I tipi di inquadrature preponderanti nel dataset sono 5 e 6 e per questo probabilmente il modello non viene addestrato in modo soddisfacente sulle altre tipologie di inquadrature da predire.

Inoltre, il modello è stato testato anche su lunghezze più piccole e più grandi ma nel primo caso le sequenze erano molto spesso prive di contenuto informativo in quanto formate solo da 5 o solo da 6, mentre nel secondo il dataset diventava troppo piccolo ed il modello non riusciva ad apprendere correttamente. Per questo è stato deciso di proporre l'algoritmo con n=5 in quanto i casi di sequenze prive di contenuto informativo erano molto pochi ed irrilevanti e il dataset aveva dimensioni soddisfacenti.

Quello che però emerge da questa analisi è il fatto che introdurre il Meccanismo di Attenzione in un'analisi di questo tipo tendenzialmente porta a dei miglioramenti dei valori di Accuracy. Inoltre, a livello computazionale e di tempi di analisi, introdurre il meccanismo di attenzione non ha portato ad un aumento del tempo del train. Questo significa che, se applicato a un dataset più grande di quello preso in considerazione, porterebbe sicuramente a dei miglioramenti dell'analisi e non ad un eccessivo tempo di addestramento del modello.

Conclusioni

Nel lavoro illustrato, è stato ampiamente evidenziato quale sia l'obiettivo della tesi, ovvero l'implementazione di una metodologia innovativa tramite l'utilizzo di Deep Learning da utilizzare nel campo cinematografico per classificare il genere cinematografico note le sequenze di tipologie di inquadrature.

Questo modello potrebbe essere applicato a diversi campi come Video Making e Streaming in quanto solo conoscendo le sequenze di tipologie di inquadrature si potrebbe identificare il genere. In particolare, potrebbe essere utilizzato nei servizi Streaming nel momento in cui è necessario definire il genere di un determinato film a supporto dei meccanismi già esistenti o nel campo dei Video Making permettendo di suggerire al registra, qualora stia girando ad esempio un film di azione, quale sia il tipo di inquadratura successivo consigliato.

Il risultato ottenuto non è ancora tale da poter pensare di applicare lo strumento nella realtà dell'industria cinematografica ma indica la giusta direzione. Inoltre, sarebbe possibile aumentare notevolmente le performance del modello, se implementato in un dataset più grande.

Infatti, nel primo modello, i valori di accuratezza di Train raggiungono il 90% e i valori di test circa il 50%. Il risultato, però, se rapportato a tutti i limiti del dataset non è da considerarsi insufficiente poiché dimostra tutte le potenzialità del modello stesso.

Dal secondo modello, invece, si evince uno dei possibili sviluppi futuri del modello stesso. Se, infatti inserito il Meccanismo di Attenzione in modelli di questo tipo, è molto probabile che le performance migliorino.

Ringraziamenti

Come conclusione di questo lavoro, desidero innanzitutto ringraziare la Professoressa Tania Cerquitelli e Bartolomeo Vacchetti per l'estrema disponibilità e gentilezza che hanno mostrato nei miei confronti e per avermi seguito in ogni passo di questo lavoro.

Inoltre, desidero ringraziare le persone che sono state parte fondamentale del mio percorso:

Ai miei genitori e a mio fratello Lorenzo, per avermi permesso di raggiungere questo traguardo credendo nelle mie potenzialità e rimanendo, in ogni occasione, sempre dalla mia parte. Avete sempre fatto per me molto di più di quello che un figlio e una sorella possa mai desiderare.

A *Elena*, la mia costante da una vita. Il mio posto sicuro nel mondo dove rifugiarmi per chiedere consigli e ricevere la risposta giusta. Sei da sempre fonte di ispirazione per me.

A *Giorgia*, uno dei regali più belli che mi ha fatto il Politecnico. La tua positività e la tua anima buona sono stati per me, essenziali in questo percorso.

A *Diego, Marta, Riccardo e Viviana* che più di essere miei amici sono la mia famiglia. Insieme abbiamo condiviso gioie, dolori, piante e risate e custodisco ogni singolo momento con tanta cura. La fortuna che ho nell'avervi accanto è per me inspiegabile, vi voglio bene.

A *me stessa*, testarda e tenace, per essere arrivata a destinazione e per avercela fatta, finalmente.

Bibliografia

[1] “Intelligenza Artificiale: cos’è, come funziona e a cosa serve?”, http://www.intelligenzaartificiale.it/#L8217Intelligenza_Artificiale_dalle_origini_ai_giorni_d8217oggi

[2] Sara Brunelli, “Intelligenza Artificiale: cos’è e cosa offre alle aziende”, https://www.cwi.it/tecnologie-emergenti/intelligenza-artificiale

[3] Paola Mello, “Che cosa fa l’Intelligenza Artificiale?”, http://disf.org/intelligenza-artificiale

[4] Fastweb, “L'Intelligenza Artificiale supera il test di Turing per la terza volta”, https://www.fastweb.it/web-e-digital/l-intelligenza-artificiale-supera-il-test-di-turing-per-la-terza-volta/

[5] Nicoletta Boldrini, “Cos’è il Machine Learning, come funziona e quali sono le sue applicazioni”, https://www.ai4business.it/intelligenza-artificiale/machine-learning/machine-learning-cosa-e-applicazioni/

[6] “L’evoluzione del Machine Learning”, https://www.sas.com/it_it/insights/analytics/machine-learning.html

[7] “I diversi apprendimenti di una macchina”, http://www.intelligenzaartificiale.it/machine-learning/

[8] Lorenzo Govoni, “Rete Neurale, Deep Learning e principali applicazioni”, https://lorenzogovoni.com/deep-learning-e-applicazioni/

[9] Elio Piccolo, “Reti Neurali”, https://areeweb.polito.it/didattica/gcia/Apprendimento_Mimetico/Lucidi/Lucidi_Reti_Neurali1.pdf

[10] Dennis Sayed, “Riassunti Reti Neurali”, http://dennis.altervista.org/pdf/neurali.pdf

[11] Nicoletta Boldrini, “I sistemi esperti e le Reti Neurali”, https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/

[12] “ImageNet, Hinton e il successo del Deep Learning”, http://www.intelligenzaartificiale.it/deep-learning/

[13] Nicoletta Boldrini, “Cos’è il Deep Learning”, https://www.ai4business.it/intelligenza-artificiale/deep-learning/deep-learning-cose/

[14] Davide Maltoni, “Deep Learning”, http://bias.csr.unibo.it/maltoni/ml/DispensePDF/9_ML_DeepLearning.pdf

[15] Francesco Pugliese e Matteo Testi, “Una panoramica introduttiva su Deep Learning e Machine Learning”, https://www.deeplearningitalia.com/una-panoramica-introduttiva-su-deep-learning-e-machine-learning/
[16] Petra Invernizzi, “Deep Learning: che cos’è, le applicazioni, gli ostacoli”, https://illuminotronica.it/deep-learning-significato-applicazioni-limiti/
[17] “Rete neurale ricorrente”, https://en.wikipedia.org/wiki/Recurrent_neural_network
[18] Denny Britz, “Recurrent Neural Networks Tutorial, Part 1 - Introduction to RNNs”, http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
[19] “Algoritmo Back-Propagation (Reti Neurali)”, https://www.okpedia.it/algoritmo_back_propagation_reti_neurali
[20] Vibhor Nigam, “Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning”, https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90
[21] Gabriele Sarti, “Cos’è una rete neurale convoluzionale?”, https://it.quora.com/Cos%E2%80%99%C3%A8-una-rete-neurale-convoluzionale
[22] Debarko De, “RNN or Recurrent Neural Network for Noobs”, https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860
[23] Christopher Olah, “Understanding LSTM Networks”, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[24] Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola, “Dive into Deep Learning”, https://d2l.ai/chapter_recurrent-neural-networks/gru.html
[25] Eu Jin Lok, “Episode 5: Building castles in the sky, or a memory palace Part 1”, https://mungingdata.wordpress.com/2018/03/21/episode-5-building-castles-in-the-sky-or-a-memory-palace-part-1/
[26] “Gated recurrent unit”, https://en.wikipedia.org/wiki/Gated_recurrent_unit
[27] Eu Jin Lok, “Episode 6: Building castles in the sky, or a memory palace Part 2” https://mungingdata.wordpress.com/2018/04/13/episode-6-grus/
[28] “Sequence Analysis”, https://en.wikipedia.org/wiki/Sequence_analysis
[29] “Sequence Alignment”, https://en.wikipedia.org/wiki/Sequence_alignment
[30] Dbdmg PoliTO, “Slide di Data Science”
[31] “I Big Data vi parlano. Li state ascoltando?”, http://italy.emc.com/microsites/cio/articles/big-data-pwf/pwf.pdf

[32] R. Sathya, Annamma Abraham, “Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification”, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.278.5274&rep=rep1&type=pdf#page=41
[33] Zengyou He , Guangyao Xu , Chaohua Sheng , Bo Xu , Quan Zou, “Reference-Based Sequence Classification”
[34] Zhengzheng Xing, Jian Pei, Eamonn Keogh, “A Brief Survey on Sequence Classification”
[35] “Introduzione al concetto di LSTM”, https://datascience.eu/it/apprendimento-automatico/comprensione-delle-reti-lstm/
[36] “5 Techniques to Prevent Overfitting in Neural Networks”, https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html
[37] “Lorenzo Govoni”, https://www.lorenzogovoni.com/5-tecniche-di-cross-validation/