POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Gestionale

Tesi di Laurea Magistrale

Classificazione delle inquadrature cinematografiche mediante una metodologia Deep Learning

Relatore/i
Prof.ssa Tania Cerquitelli

Candidato
Francesco Notarangelo

Anno Accademico 2019/2020

Indice

INTRODUZIONE

Questa tesi è nata dal mio felice incontro con la Professoressa Tania Cerquitelli, docente del corso di laurea magistrale in ingegneria gestionale del Politecnico di Torino - che è il relatore del mio elaborato finale - e con il Dott. Ing. Bartolomeo Vacchetti, titolare di una borsa di studio per attività di ricerca presso lo stesso Politecnico. Con il loro qualificato supporto ho affrontato una sfida che mi ha da subito affascinato per l'obiettivo, arduo ma esaltante, da raggiungere: utilizzare tecniche di apprendimento automatico per la classificazione di immagini da applicare al campo creativo e, in particolare, al mondo del cinema. Più specificamente, il progetto è stato avviato con l'intento di individuare e sperimentare, attraverso il ricorso all'intelligenza artificiale, una metodologia in grado di facilitare in misura notevole il lavoro dell'editor dei video che, quando procede al montaggio delle riprese cinematografiche, si trova a dover gestire, nella maggior parte dei casi, una considerevole mole di dati disorganizzati. Egli dispone, infatti, di una serie di file che, pur essendo archiviati nella stessa cartella, non sono ordinati secondo le sequenze del film e, quindi, richiedono, nel processo di editing, operazioni lunghe e complesse di risistemazione. L'editor non conosce il contenuto all'interno di ogni file video e, pertanto, è costretto a spendere tempo e risorse per cercare, di volta in volta, quello che serve oppure ad ordinarli manualmente impiegando giorni o settimane per completare il lavoro.

L'utilizzo di un sistema di apprendimento automatico permette invece di far acquisire alla macchina le informazioni e conoscenze necessarie per procedere a diversi tipi di classificazione accorciando i tempi e rendendo l'attività di montaggio, oltre che molto più celere, anche efficace e funzionale nell'organizzazione.

Con queste premesse e con l'auspicio di fornire un utile contributo alla implementazione dell'utilizzo dell'I.A. nelle attività della vita quotidiana, mi sono avventurato lungo un cammino da me sinora inesplorato che mi ha portato ad approfondire metodi e sistemi di analisi per i quali avevo sempre nutrito, nel corso dei miei studi, un particolare interesse ma che non avevo avuto modo di sperimentare sul campo.

Anche i tentativi non andati a buon fine e i momenti di scoramento hanno arricchito il mio bagaglio di esperienza e di conoscenze stimolandomi a cercare soluzioni correttive o a percorrere nuove strade e accrescendo, di giorno in giorno, l'entusiasmo per ogni conquista e il desiderio di raggiungere il traguardo prefissato. Mi ha sostenuto, in questo impegno, la mia antica passione per il mondo del cinema che ho coltivato negli anni, partecipando, tra l'altro, come co-conduttore, al programma "Zapping" trasmesso da "Ondequadre", la radio del Politecnico di Torino.

Lavorando alla tesi ho potuto esplorare l'universo "sommerso" della cinematografia, poiché, partendo dalle immagini e dalle inquadrature sono entrato nel vivo dell'attività di backstage che è dietro la costruzione di un film e ho scoperto quanto le interazioni con il mondo dell'informatica possano semplificarne i meccanismi rendendola più veloce e meno faticosa.

La tesi si articola in sei capitoli: i primi quattro costituiscono una premessa necessaria ad illustrare successivamente, entrando nel vivo dell'analisi e della sperimentazione svolta, le metodologie e le tecniche implementate per giungere al risultato auspicato, ovvero quello di classificare immagini e video attraverso la creazione di diversi set di dati e il successivo addestramento di alcuni modelli di Deep Learning presenti allo stato dell'arte.

Infatti, nel primo capitolo, vengono introdotte e brevemente illustrate le diverse tipologie di machine learning, mentre il secondo è dedicato all'esame dell'architettura, della metodologia di apprendimento e delle principali funzioni delle reti neurali per passare poi, nel terzo capitolo, ad un approfondimento sulle reti neurali convoluzionali (CCN) e sull'ensemble learning, in quanto entrambi gli strumenti sono stati utilizzati nell'analisi sperimentale su cui si basa l'elaborato.

Il quarto capitolo contiene un rapido excursus della classificazione delle inquadrature cinematografiche, anche per delineare compiutamente quali di esse, tra le diverse categorie esistenti, sono state prescelte per la sperimentazione.

Inoltre, sono stati esaminati, nello stesso capitolo, alcuni esempi di applicazione della loro classificazione, mediante l'utilizzo del deep learning.

Completato l'esame dello scenario entro il quale si è mossa l'attività sperimentale, nel quinto e nel sesto capitolo sono state descritte le varie fasi dell'analisi, dalla creazione e l'esplorazione dei dataset adoperati tramite diverse tecniche (Image Segmentation, Hypercolumns extraction) alle strumentazioni utilizzate (VVG16,

transfer learning, stacking ensemble learning) sino alla descrizione dei risultati raggiunti, esaminando le principali componenti dell'analisi stessa e analizzando le performance ottenute tramite metodi di explanation AI.

Le conclusioni riguardano invece una comparazione tra lo "stato dell'arte" cui si è pervenuti attraverso l'uso delle tecnologie illustrate e le possibili prospettive future, partendo dal presupposto che l'indagine portata avanti può costituire non solo il punto di partenza per una applicazione in svariati altri campi creativi, ma anche la premessa per una ulteriore implementazione, attraverso il ricorso ad altre metodologie innovative fra cui, ad esempio, il federated learning, al quale, proprio in vista di potenziali evoluzioni sperimentali, è stato dedicato l'Annex inserito a margine dell'elaborato.

CAPITOLO 1: INTELLIGENZA ARTIFICIALE E MACHINE LEARNING

1.1 **La rivoluzione digitale attraverso l'intelligenza artificiale**

Nel libro "*The second machine age*[1]" gli autori intravedevano profilarsi all'orizzonte una importante crescita dell'economia mondiale trainata dalle "macchine intelligenti", ovvero quelle che si avvalgono dei processori di ultima generazione, dell'intelligenza artificiale e dei sistemi di comunicazione integrata.
In realtà la tecnologia digitale, gli sviluppi dell'informatica e della telematica e le implementazioni nell'utilizzo dei Big Data hanno prodotto una vera e propria rivoluzione che sta coinvolgendo tutti gli aspetti della vita quotidiana poiché è cambiato radicalmente il modo di produrre, elaborare, raccogliere e scambiare informazioni.
L'attuale congiuntura storica e socioeconomica è infatti caratterizzata dalla crescente digitalizzazione che ci consente di essere sempre "connessi", di socializzare al di là delle distanze fisiche e di ottenere informazioni in tempo reale.
Nel contempo alcune professioni tradizionali sono divenute desuete, mentre altre sono emerse e si stanno a poco a poco imponendo sullo scenario globale.
Siamo entrati a pieno titolo in quella nuova fase del processo di industrializzazione che già nel 2011, con un termine coniato durante la fiera di Hannover, è stata definita industria 4.0, un percorso innovatore che si fonda proprio sull'interconnessione, sulla velocità delle informazioni e sull'intreccio di tecnologie sempre più innovative.

[1] The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies , Erik Brynjolfsson e Andrew McAfee, 18 febbraio 2014

Le sfide di questa nuova era si basano soprattutto su un approccio orientato a valorizzare le potenzialità dell'A.I., l'Intelligenza Artificiale.

L'A.I. mostra uno sviluppo costante ormai da più di mezzo secolo e si avvia verso una nuova fase di crescita, supportata dai progressi tecnologici scaturiti dall'uso di internet, della enorme mole di dati disponibili, delle reti di sensori, del super computing, tanto da porsi quale componente indispensabile della strategia di automazione di una qualsiasi organizzazione.

L'intelligenza artificiale si distingue per essere la più "disruptive" tra le nuove tecnologie grazie ai significativi risultati raggiunti nel corso degli anni in molti ambiti di applicazione e alle incoraggianti prospettive di ulteriori e più pervasive applicazioni.

Per questo non può più essere valutata come un'opzione ma va vista invece come un'esigenza, una opportunità di vantaggio competitivo.

Basti pensare all'aiuto fornito alle imprese per ottimizzare la qualità del lavoro, aumentare fatturati e profitti, favorire l'acquisizione di nuovi clienti, migliorare l'efficienza e la produttività. Ma non solo. L'A.I. costituisce anche e soprattutto uno strumento per ampliare ed espandere l'intelligenza umana, offrendo alla mente conoscenze contestuali alle quali da sola non riuscirebbe ad accedere o che non potrebbe facilmente elaborare. Inoltre, essa può abilitare i processi robotici all'automiglioramento e all'autocorrezione e presenta una enorme utilità per impostare un applicativo senza che sia necessaria l'interazione umana. Infine, va evidenziato che l'A.I., in epoca recente, è stata utilizzata nel riconoscimento di chat e voce nei dispositivi mobili e nei servizi on line ed è stata sperimentata con successo nell'analisi delle immagini in ambiti di sicurezza, automazione industriale e, da ultimo, nella guida autonoma.

Per dare una definizione il più possibile aderente al potenziale insito nel concetto di Intelligenza Artificiale non basta ricorrere al solo linguaggio informatico. L'accezione secondo cui l'AI. è classificata come "la disciplina che racchiude le teorie e le tecniche pratiche per lo sviluppo di algoritmi che consentano alle "macchine" di mostrare attività intelligente"[2] appare infatti riduttiva. Ampliandone invece la portata e le finalità potremmo definirla partendo dal funzionamento del cervello umano e individuandone le funzioni. In questo senso una AI deve saper

[2] https://www.cospe.org/wp-content/uploads/2019/07/03_dossier_INTELLIGENZA ARTIFICIALE_080719-1.pdf

compiere alcune azioni tipiche dell'uomo, ovvero agire (come farebbe un essere umano), pensare utilizzando funzioni cognitive proprie della mente umana e pensare razionalmente, con la stessa logica dell'uomo.

In sintesi, potremmo affermare che l'AI è "una disciplina, sia scientifica che ingegneristica, che si occupa di realizzare macchine pensanti, ovvero macchine e programmi che possano risolvere, in modo del tutto autonomo, vari problemi con il ragionamento. Tale ragionamento assumerà le vesti della riflessione di tipo logico, intelligente e soprattutto razionale"[3].

Da un punto di vista più squisitamente tecnologico L'AI utilizza un'infrastruttura in grado di ricevere input e restituire output dove gli input sono i dati immessi nel sistema per essere processati e analizzati dall'algoritmo (detto machine learning) oppure dalla rete neurale (deep learning) che, in base all'obiettivo prefissato, li rielabora e genera un output. Inoltre, l'Intelligenza Artificiale assolve a diversi livelli funzionali: la comprensione, il ragionamento e l'apprendimento.

Essa infatti mediante la simulazione della capacità cognitiva di mettere in correlazione dati ed eventi, riesce a riconoscere voce, testi, immagini, video e a ricavarne informazioni (comprensione), quindi, attraverso un processo logico e utilizzando algoritmi, collega tutte le informazioni raccolte (ragionamento), successivamente, con diverse tecniche di apprendimento automatico, impara a svolgere determinate funzioni. Infine, può interagire con l'uomo (ciò avviene, ad esempio, nei sistemi di Natural Language Processing).

La comunità scientifica ha posto una distinzione fondamentale alla base del concetto di Intelligenza artificiale al fine di delinearne con maggiore precisione i limiti e le reali capacità operative. Lo studioso John Searle ha per primo individuato le differenze tra Intelligenza artificiale debole (*weak intelligence)* e Intelligenza Artificiale forte (*strong intelligence*) evidenziando che la prima mira a realizzare sistemi che possono svolgere alcune funzioni umane simulando il comportamento dell'uomo, senza però mai eguagliarlo o superarlo[4]. Dal punto di vista pratico L'AI, debole opera indagando su casi simili, confrontandoli ed elaborando varie soluzioni per poi scegliere quella più razionale e più congrua. Ciò significa che la macchina agisce con precisione ed efficienza ma non crea processi mentali, semplicemente

[3] www.intelligenzaartificiale.it/intelligenza-artificiale-forte-e-debole/

[4] www.morpheos.eu/2018/11/26/intelligenza-artificiale-forte-e-intelligenza-artificiale-debole

raccoglie i dati dell'esperienza acquisita come farebbe la mente umana senza che sia necessario comprendere totalmente i processi cognitivi dell'uomo. In sintesi, l'AI debole è deputata alla risoluzione dei problemi (attraverso programmi matematici di problem-solving si sviluppano funzionalità per consentire alle macchine di valutare ipotesi a livello empirico e di prendere decisioni) ma non ha la capacità di pensare autonomamente e ha bisogno della presenza dell'uomo.

L'AI forte viene invece definita come quella metodica in grado di conferire alla macchina una capacità cognitiva non distinguibile da quella umana. Alcuni scienziati a tal proposito hanno parlato di "sistemi sapienti" o addirittura "coscienti di sé" [5], cioè tali da poter sviluppare una intelligenza propria senza emulare processi mentali e cognitivi simili a quelli umani. Alla base di tale sistema vi sarebbero alcuni fattori fondamentali: la logica matematica che rappresenta l'intero scibile umano; il ragionamento e la dimostrazione automatica del problema; l'analisi del linguaggio, elemento indispensabile affinché la macchina possa comprendere le espressioni linguistiche umane e la pianificazione, attraverso algoritmi.

Questa seconda visione dell'intelligenza Artificiale si riferisce a un tipo di intelligenza in grado di sostituire l'essere umano nella sua interezza, incluse tutte le sue molteplici abilità e di porsi addirittura in una posizione di supremazia rispetto all'uomo.

Tuttavia, nessuna macchina sinora ha superato il test del matematico Alan Turing secondo cui se un uomo, chiuso in una stanza, pone delle domande ad un calcolatore attraverso una tastiera e non riesce a comprendere se dall'altra parte della stanza le risposte gli vengono fornite da un essere umano o da una macchina, allora siamo in presenza di un calcolatore intelligente[6]. Sembra invece più convincente nel mondo della scienza la teoria del filosofo John Searle il quale, con un altro test[7] , ha voluto dimostrare che se anche una macchina dovesse dare l'impressione di essere in grado di pensare intrattenendo con noi una discussione, in realtà essa starà solo eseguendo una serie di operazioni guidate in quanto non acquisirà mai il "contenuto mentale", un concetto simile alla "coscienza". Searle, fautore della concezione cognitiva,

[5] Algoritmi per l'intelligenza artificiale (versione in eBook) – Capitolo 1 - Progettazione dell'algoritmo - Dati e Machine Learning - Neural Network - Deep Learning di Roberto Marmo

[6] La macchina di Turing può esser definita "un modello matematico computazionale che definisce una macchina astratta in grado di manipolare stringhe di simboli secondo delle regole sintattiche precise"

[7] Argomento della Stanza Cinese, Minds, Brains and Programs ,) John R. Searle, rivista *Behavioral and Brain Science, 1980*

contrapponendosi a quella computazionale di Turing, ha dunque sottolineato che le macchine pensanti dovrebbero innanzitutto essere capaci di duplicare i processi biologici che stanno alla base della nostra cognizione. Alla luce di questa visione i normali computer non possono che essere considerati macchine di Turing approssimate[8].

Questo affascinante dibattito nell'era moderna è stato superato dalla teoria secondo cui una macchina potrà essere definita intelligente solo quando sarà in grado di riprodurre il funzionamento del cervello a livello cellulare.

Non a caso Eliezer Yudkowsky del Machine Intelligence Research Institute ha detto; "L'intelligenza artificiale non è scienza acquisita; è scienza di frontiera, non da manuale"[9]

In realtà siamo ancora lontani dal poter ritenere una AI paragonabile all'intelligenza umana, tuttavia gli studiosi, forti dei promettenti risultati finora ottenuti non solo nell'ambito del business e del lavoro in genere, ma anche nell'impatto del consumatore con tali tecnologie (si pensi alle Over the top come Google e Facebook o a Siri di Apple o ad Amazon) ritengono che quando "saremo in grado di chiarire in termini computazionali quali possono essere le differenze negli esseri umani tra conscio e inconscio, codificarlo nei computer non può essere così difficile".[10] Con queste premesse i ricercatori hanno esplorato il cervello umano cercando di definire una road map a più livelli (C0, C1,C2) utile per progettare una AI dotata di coscienza. Il primo livello riguarda le operazioni inconsce come il riconoscimento del viso o del linguaggio, il secondo la capacità di assumere decisioni dopo aver attinto a una vasta gamma di pensieri e dopo aver preso in considerazione una serie di possibilità, il terzo la capacità di monitorare i propri pensieri, ovvero di essere auto-consapevoli.

Resta perciò sul tappeto la dibattuta questione-che vede interessati non solo gli esperti della comunità scientifica ma anche filosofi, economisti e sociologi- sulle

[8] Sarle diceva; *"Potrebbe una macchina pensare?" La mia opinione è che solo una macchina possa pensare, e, in verità, che solo tipologie speciali di macchine, cioè cervelli e macchine che abbiano gli stessi poteri causali dei cervelli".*

[9] Artificial Intelligence as a Positive and Negative Factor in Global Risk EliezerYudkowsky MachineIntelligenceResearchInstitute, 2008

[10] AI Artificial Intelligence: Come è nata, come funziona e come l'Intelligenza ...Di Nicoletta Boldrin, 2018

implicazioni etiche e sociali che deriverebbero dalla possibilità di un computer di sviluppare una coscienza umana[11].

1.2 **Il Machine Learning**

l'Intelligenza Artificiale sotto il profilo tecnologico e metodologico è il modello di apprendimento con cui l'intelligenza diventa abile in un compito o azione. In base ai diversi modelli di apprendimento possiamo operare una distinzione tra Machine Learning e Deep Learning.

Il machine learning o apprendimento automatico è una branca dell'intelligenza artificiale che prevede diverse modalità, tecniche e strumenti per essere realizzata. In particolare, si tratta di "meccanismi che permettono a una macchina intelligente di migliorare le proprie capacità e prestazioni nel tempo**"**. La macchina impara a svolgere determinati compiti e, attraverso l'esperienza[12], migliora le proprie capacità di risposta e le proprie funzioni. Alla base dell'apprendimento automatico ci sono una serie di algoritmi che, partendo da nozioni primitive, saranno in grado di orientarsi verso una specifica decisione piuttosto che un'altra o di compiere azioni apprese nel tempo.

In sostanza il computer impara ad eseguire una *task* senza che sia stato espressamente programmato per eseguirla. Il machine learning infatti utilizza la costruzione di algoritmi che permettono un apprendimento di informazioni partendo dai dati disponibili e, nello stesso tempo, consentono anche di predire nuove informazioni sfruttando quelle già apprese. Viene quindi costruito un modello che, partendo da osservazioni, acquista automaticamente una capacità predittiva ed è perciò in grado di fornire nuovi dati e svolgere nuove funzioni senza che vi sia stato un intervento di programmazione a monte.

---

[11] Non è un caso che la Commissione Europea abbia tracciato, nel 2019, apposite "linee guida etiche per un'intelligenza artificiale affidabile". con l'obiettivo di collocare l'intelligenza artificiale in una dimensione etica**,** che non possa prescindere dal rispetto dei diritti fondamentali delle persone, mirando a rafforzarne le capacità e non anche a sostituirsi ad esse. https://ec.europa.eu/commission/presscorner/detail/it/IP_19_1893

[12] Bishop, C. M. (2006), Pattern Recognition and Machine Learning, Springer, New York, NY: Springer, 2006. - 738 p

Oggi le nozioni di apprendimento automatico, intelligenza artificiale e macchine intelligenti sono entrate a far parte del linguaggio comune degli "addetti ai lavori" ma per giungere ai risultati attuali il processo è stato lungo e complesso, alternando fasi di felice sperimentazione a momenti di scoramento e scetticismo. I primi esperimenti si devono a matematici e statistici che negli anni Cinquanta pensarono di ricorrere ai metodi probabilistici per realizzare macchine che potessero scegliere tra opzioni diverse considerando le probabilità del verificarsi di un evento.

Il termine machine learning è stato coniato dall'informatico statunitense Arthur Samuel e ripreso successivamente dall'informatico Tom Mitchell che ne ha dato una definizione formale e ancora attuale: "*Un programma apprende da una certa esperienza E se nel rispetto di una classe di compiti T, con una misura della prestazione P, la prestazione P misurata nello svolgere il compito T è migliorata dall'esperienza E".*

In altre parole, un modello di machine learning apprende costantemente attraverso l'esperienza e le regole non vengono definite preventivamente in modo rigido dal programmatore in quanto quest'ultimo si limita a definire delle "feature" (caratteristiche) di interesse e poi lascia alla macchina il compito di imparare attraverso l'analisi dei dati disponibili e di trovare autonomamente il modo per raggiungere il risultato operando generalizzazioni, classificazioni e riformulazioni. Da ciò scaturisce la differenza nelle prestazioni del ML rispetto ad un sistema tradizionale che è invece basato su regole predeterminate e che, pertanto, esegue i propri compiti sempre nello stesso modo, senza mai migliorare mediante l'apprendimento.

Più precisamente potremmo dire che un algoritmo classico, una volta codificato con determinate regole, quando riceve i dati come input, restituisce risposte come output.

Diversamente un algoritmo di machine learning riceve dati come input ma restituisce regole come output, ovvero è capace di apprendere dai dati.

Rispetto ad un approccio tradizionale che consiste nell'individuare una funzione specifica in base alla quale un certo input produrrà sempre un determinato output, nel machine learning vengono utilizzati algoritmi generici, di tipo matematico e statistico, che, dopo aver ricevuto una serie di dati, attraverso una fase di addestramento (training), cui seguono la valutazione dei risultati e l'ottimizzazione

dei parametri, ricavano la funzione autonomamente e questa non sempre è nota al programmatore.
Volendo ulteriormente esemplificare poniamo che la funzione sia *y=f(x)*: nell'approccio tradizionale il programmatore descrive alla macchina nel dettaglio il funzionamento di *f(x)*, nel machine learning invece il modello è tale da individuare, analizzando una serie differente di dati, il valore più probabile di y, dunque è in grado di ricavare da solo la funzione *f(x)* .
Le potenzialità di tale modello stanno soprattutto nella capacità di analizzare una enorme mole di dati considerando innumerevoli variabili e di individuarne caratteristiche comuni oppure di elaborare tendenze.

Nel caso delle immagini- che nel presente lavoro riveste particolare interesse-lo scopo può essere quello di realizzare sistemi capaci di riconoscere e classificare con accuratezza un dataset con un numero elevato di classi.

1.3 **Le tipologie di Machine Learning principali**

Esistono diverse modalità di apprendimento automatico che differiscono fra loro in base agli algoritmi utilizzati e agli obiettivi da raggiungere.
I modelli su cui si basa il sistema di machine learning vengono comunemente classificati in tre categorie: "*Supervised Learning"*, "*Unsupervised Learning"* e *"Reinforcement Learning"*.

1.3.1 *Supervised Learning*

"Nell'apprendimento supervisionato il lavoro di risoluzione viene lasciato al computer. Una volta compresa la funzione matematica che ha portato a risolvere uno specifico insieme di problemi, sarà possibile riutilizzare la funzione per rispondere a qualsiasi altro problema similare"[13]

[13] Adam Geitgey, "Machine Learning is Fun!", Maggio 2014

In questo caso al computer vengono erogati set di dati[14] come input ma anche informazioni sul risultato da raggiungere, ovvero vengono forniti alla macchina esempi di input e di output in modo che impari a collegarli fra loro ed elabori una regola per stabilire tale nesso, regola che poi riutilizzerà ogni volta che dovrà svolgere compiti simili. Il computer sarà cioè posto in grado di mappare gli input negli output e, pertanto, una volta compresa la funzione matematica alla base della risoluzione di un insieme di problemi, sarà capace, utilizzando quella funzione, di individuare la soluzione a tutti i problemi similari.

Difatti il sistema informatico farà tesoro dell'esperienza acquisita sulla base dei modelli e degli esempi forniti [15]dal programmatore e costruirà un vero e proprio database costituito da informazioni ed esperienza cui attingerà per restituire la risposta più adeguata al problema sottoposto. Da quanto esposto appare evidente che questo tipo di apprendimento è già preconfezionato e che la macchina si limita a scegliere la risposta migliore allo stimolo ricevuto.

Detto sistema consente alla macchina di effettuare ipotesi induttive, ovvero di scansionare problemi specifici per arrivare alla risoluzione di problemi di carattere generale. Per questo il suo impiego è diffuso in ambito medico o per l'identificazione vocale.

1.3.2 *Unsupervised Learning*

Nell'apprendimento non supervisionato[16] invece non vengono fornite alla macchina informazioni codificate e strutturate, quindi essa non viene posta nelle condizioni di conoscere i risultati attesi a secondo della scelta effettuata. Pertanto, non può basarsi sull'esperienza ma deve prima catalogare tutte le informazioni ricevute, organizzarle, impararne il significato, capire come utilizzarle e a quali risultati portano in base alla scelta effettuata. Il computer dispone infatti soltanto di dati di input senza alcun output atteso[17]. In questo caso il compito dell'algoritmo è quello di individuare pattern nascosti all'interno dei dati disponibili oppure di estrapolare

[14] https://www.ai4business.it/intelligenza-artificiale/machine-learning/machine-learning-cosa-e-applicazioni/

[15] https://www.filodiritto.com/machine-learning

[16] https://www.beantech.it/blog/articoli/machine-learning-cosi-le-macchine-imparano-ad-migliori/

[17] https://www.deeplearningitalia.com/a-general-introduction-to-learning-methods-2/

le caratteristiche peculiari dei dati (feature) al fine di eseguire un altro task di machine learning.

1.3.3 *Reinforcement Learning*

Infine, l'apprendimento con rinforzo[18] prevede che la macchina sia dotata di sistemi e strumenti di supporto tali da consentirle di interagire con un ambiente dinamico e di comprenderne le caratteristiche in modo da effettuare scelte per adattarsi ad esso nel modo migliore. Mentre il computer esplora il dominio del problema e cerca la soluzione più idonea gli vengono forniti dei feedback positivi o negativi affinché, imparando anche dagli errori, raggiunga l'obiettivo a seguito di una routine di apprendimento sostanzialmente basata su punizioni e ricompense. Gli strumenti di supporto che vengono forniti alla macchina possono essere i più vari come avviene in uno dei casi più comuni di utilizzo di tale sistema, ovvero la guida automatica senza pilota, In tal caso essa dispone di sensori, telecamere, GPS e interagisce con l'ambiente circostante che le fornisce una serie di dati di input.

Il Machine Learning viene utilizzato soprattutto per risolvere tre ordini di problemi[19] : la classificazione, laddove sia necessario decidere a quale categoria appartiene un determinato tipo di dato, la regressione, quando, avendo noto il valore attuale di un dato, è necessario prevederne il valore futuro (ad esempio per la quotazione delle azioni di una società) e il clustering o raggruppamento quando la finalità è quella di raggruppare dati con caratteristiche simili (impiegato dalle aziende per individuare il target potenziale della clientela).
Nell'apprendimento supervisionato si ricorre alla classificazione per sviluppare modelli predittivi e perché i dati possono essere divisi in categorie, gruppi o classi specifiche (ad esempio per l'imaging medico o per il riconoscimento vocale) ma anche alle tecniche di regressione dove i segnali di output sono dei valori continui come nelle variazioni di temperatura o nelle fluttuazioni del consumo di energia.
Nell'apprendimento supervisionato si utilizza il clustering tra le cui applicazioni più comuni figurano l'analisi della sequenza genetica, le ricerche di mercato e il riconoscimento di oggetti.

[18] https://www.hudi.it/2019/10/02/i-tre-tipi-di-machine-learning/

[19] https://www.4next.eu/blog/intelligenza-artificiale-definizione/

L'apprendimento semi supervisionato è un modello ibrido[20] nel quale alla macchina viene fornito un set di dati incompleti: alcuni di essi sono dotati di esempi di output (etichettatura come nell'apprendimento supervisionato), altri invece sono privi di tali esempi (non sono etichettati). Lo scopo di questo sistema di apprendimento automatico è quello di utilizzare, per l'allenamento, una piccola quantità di dati codificati insieme a una grande quantità di dati senza etichetta. Ciò si verifica quanto l'etichettatura dei dati è molto costosa oppure se si dispone di un flusso costante di dati. Lo scopo è sempre quello di individuare regole e funzioni per risolvere problemi, nonché modelli utili a raggiungere determinati obiettivi.
La scelta tra i due approcci[21] (con o senza supervisione) viene spesso operata per tentativi. È ovvio però che essa dipende anche dal tipo di dati utilizzati, dal loro formato e dai risultati che si desidera ottenere (quale tipo di informazioni e per quale scopo).
Ad esempio, per ottenere previsioni su una variabile continua oppure operare una classificazione (variazione della temperatura, valore di un'azione sul mercato, identificare una targa da una webcam) l'apprendimento supervisionato appare il metodo più adatto mentre conviene scegliere quello non supervisionato in presenza di dati contenenti un ordinamento o un raggruppamento netto e chiaramente identificabile in modo da poterli confrontare tra loro individuando similitudini e differenze (rilevamento di anomalie per identificare attività dannose nella rete di un'organizzazione).

1.4 **Deep Learning**

Il Deep Learning, termine anglossasse che letteralmente significa apprendimento profondo, è una sottocategoria del Machine Learning che si avvale di particolari modelli di apprendimento su più livelli di rappresentazione dell'informazione (da cu il termine *"Deep"*) costruiti sulla base della struttura del cervello biologico e quindi del funzionamento dell'intelligenza umana (come le reti neurali artificiali,

[20] https://www.impresacity.it/approfondimenti/20434/i-modelli-del-machine-learning.html
[21] https://it.mathworks.com/discovery/machine-learning.html

lungamente analizzate nel capitolo successivo). Questi modelli devono essere alimentati con dati in input attraverso i quali riescono ad "apprendere" come raggiungere un obbiettivo utilizzando un algoritmo di addestramento.

Dal punto di vista scientifico il Deep Learning è indicato come l'apprendimento di dati non forniti dall'essere umano ma acquisiti tramite algoritmi di calcolo statistico, infatti può essere inserito nella famiglia delle metodologie di Machine Learning che si basano sull'acquisizione di rappresentazioni di dati e non sull'esecuzione di compiti specifici come molti dei modelli che sono parte del Machine Learning.

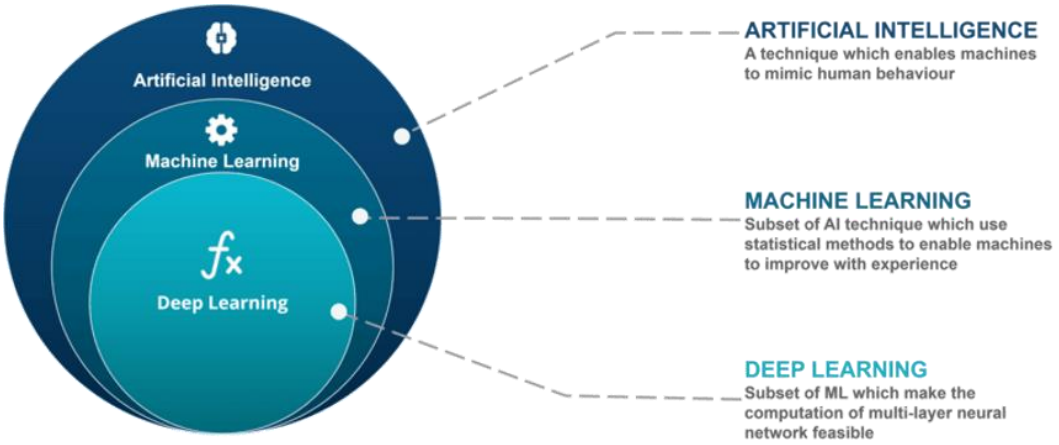


Figura 1.2: Rappresentazione della relazione tra Intelligenza Artificiale, Machine Learning e Deep Learning

Lo scopo degli algoritmi di apprendimento profondo è quello di replicare il funzionamento del cervello umano comprendendo il percorso che le informazioni compiono al suo interno e il modo in cui esso interpreta le immagini e il linguaggio naturale. Perciò le architetture di Deep Learning hanno trovato grande applicazione nella classificazione delle immagini (di specifico interesse nell'ambito del presente elaborato). Proprio in quest'ultima applicazione possiamo cogliere le differenze sostanziali tra machine learning (inteso quale apprendimento superficiale) e apprendimento profondo. Infatti, un flusso di lavoro di machine learning viene avviato con l'estrazione manuale delle feature significative dalle immagini, quindi con le feature estratte si crea un modello che categorizza gli oggetti nell'immagine. Diversamente nel deep learning l'estrazione delle feature dalle immagini avviene automaticamente e viene eseguito un apprendimento end to end nel quale una rete apprende in via autonoma come elaborare dati e svolgere un'attività.

In sostanza, l'apprendimento di dati nel deep learning [22]avviene senza l'intervento dell'uomo in quanto la macchina li apprende grazie all'utilizzo di algoritmi di calcolo statistico. Come già accennato il deep learning è un metodo molto utilizzato per effettuare l'identificazione di oggetti attraverso modelli come le reti neurali convoluzionali (che saranno oggetto di separata trattazione). Tali modelli consentono alla macchina di imparare automaticamente le caratteristiche di un oggetto e quelle che lo differenziano da altri, quindi di riconoscerlo analizzando migliaia di immagini addestrate. A tal fine esistono due approcci[23]: l'addestramento da zero e il transfer learning. Il primo richiede la raccolta di un set di dati di grandi dimensioni. È inoltre necessario progettare una architettura di rete che abbia la capacità di apprendere le feature e il modello. L'utilizzo di una gran mole di dati richiede ovviamente un certo lasso di tempo che può variare da alcuni giorni ad alcune settimane, perciò l'addestramento da zero non è di uso comune.

Il transfer learning [24]è invece l'approccio più utilizzato perché consente di affinare un modello già addestrato: in una rete preesistente come, per esempio, GoogleNet, si inseriscono nuovi dati che contengono classi in precedenza sconosciute allo scopo di poter svolgere una nuova attività. Ciò comporta alcuni vantaggi rispetto all'addestramento da zero, ossia la possibilità di riutilizzare il comportamento di una rete neurale esistente e quindi pre-addestrata andando a ridefinire solo gli ultimi strati di classificazione, nonché di limitare l'elaborazione ad un numero notevolmente inferiore di parametri con conseguente riduzione dei tempi per il raggiungimento del risultato (tale approccio verrà largamente approfondito nei capitoli successivi, essendo di fondamentale importanza per l'elaborato e utilizzato durante tutta l'analisi) .

Il Deep Learning deve il suo recente successo al progresso tecnologico e digitale del campo dell'informatica raggiunto nell'ultimo decennio. La grande produzione di dati dell'era dei Big Data dovuta alle interazioni digitali quotidiane e all'IoT (Internet Of Things)[25], soprattutto in campo aziendale, ha permesso di semplificare

[22] https://www.intelligenzaartificiale.it/deep-learning/

[23] https://it.mathworks.com/discovery/deep-learning.html

[24] https://www.developersmaggioli.it/blog/il-transfer-learning/

[25] "Per Internet of Things (IoT) o Internet delle Cose si intende quel percorso nello sviluppo tecnologico in base al quale, attraverso la rete Internet, potenzialmente ogni oggetto dell'esperienza quotidiana acquista una sua identità nel mondo digitale. Come detto, l'IoT si basa sull'idea di oggetti "intelligenti" tra loro interconnessi in modo da scambiare le informazioni" https://blog.osservatori.net/it_it/cos-e-internet-of-things

l'allenamento dei modelli con dataset sempre più vasti; inoltre l'introduzione di tecnologie come le GPU (Graphic Processing Unit), unità computazionali di ultima generazione per l'elaborazione dati, ha reso questo processo estremamente più rapido ed efficiente.

Così descritto il Deep Learning può sembrare uno strumento futuristico che richiama i film di Steven Spielberg. In realtà esso è già largamente presente nella vita di tutti i giorni e ci permette di effettuare attività quotidiane come le ricerche basate sul testo o sulle immagini, di creare filtri antispam, di effettuare traduzioni linguistiche ed è largamente applicato anche in diversi campi della medicina, ad esempio per l'individuazione di noduli cancerosi in immagini di tomografie polmonari.

Nel 2012, data epocale per il deep learning, il contest ImageNet presentò i risultati di un esperimento condotto nel settore del riconoscimento visivo. Lo scienziato Hinton mostrò che, grazie a questa nuova tecnologia, la macchina era in grado di distinguere uomini e animali confrontandoli con milioni di altre immagini, senza che l'uomo dovese effettuare alcun preventivo intervento di codificazione.

Qualche esempio pratico dell'applicazione del deep learning consente di comprenderne ancora meglio l'utilità e la portata.

Colorazione automatica di immagini in bianco e nero: gli algoritmi di deep learning sono in grado di comprendere il contesto delle immagini in bianco e nero e di cominciare a colorare laddove è necessario. Mediante l'architettura di una rete neurale, il software di Intelligenza Artificiale è in grado di attraversare un numero di immagini elevato su un database e di trovare il tono adeguato a adattarsi a ogni immagine. Alcuni vedono questo approccio particolarmente indicato per colorare fotogrammi fissi di film in bianco e nero.

Aggiunta di suoni a video muti: sfruttando l'addestramento profondo i fotogrammi video vengono associati ad un data base di suoni già registrati in modo da individuare il suono più confacente a quello che sta accadendo nella scena del video e riprodurlo.

Generazione automatica di didascalie di immagini: tale risultato è stato raggiunto sfruttando i migliori modelli ottenuti per la classificazione e il rilevamento di oggetti nelle fotografie. Infatti, le etichette rilevate e generate per tali oggetti possono essere utilizzate per creare una didascalia delle immagini. La stessa tecnica è stata successivamente applicata anche ai video.

Capitolo 2: RETI NEURALI, APPRENDIMENTO E TRANSFER LEARNING

2.1 **Introduzione alle Reti Neurali**

Per comprendere meglio il tema del Deep Learning e approfondire le tecnologie di cui si è fatto uso per la redazione della presente tesi è necessario introdurre lo strumento più potente utilizzato in questo campo, ovvero le reti neurali artificiali.
Le reti neurali sono particolari algoritmi di Machine Learning che generano modelli ispirati al funzionamento delle reti neurali biologiche e come queste, sono composti da un insieme di unità computazionali (neuroni) interconnesse tra loro e suddivise in gruppi sequenziali di stratificazione denominati "*layer"* in modo da formare una rete. Questi modelli attraverso una fase di "addestramento" specifica e in base alla tipologia di architettura (dipendente dalle interconnessioni tra neuroni) permettono di svolgere determinate task e risolvere problemi ingegneristici di vario genere come ad esempio la classificazione delle immagini (argomento del presente lavoro o l'elaborazione del linguaggio naturale.
Le reti neurali che permettono di eseguire il Deep Learning sono costituite da un numero elevato di layer e per questo motivo sono definite "Deep Neural Network". Ogni strato, partendo da quello di input e passando per una serie di layer nascosti (hidden layer), elabora le informazioni provenienti dallo strato precedente e le passa in input allo strato successivo fino a giungere all'output layer. Le connessioni tra i neuroni appartenenti ai diversi strati si basano sul processo parallelo distribuito delle informazioni (in inglese *PDP – Parallel Distributed Processing*[26]) correlato al concetto, in campo medico, di scienze cognitive. Le reti neurali artificiali esattamente come il cervello biologico elaborano tutti i dati ricevuti in input parallelamente e ridistribuiscono le informazioni acquisite su tutta la rete e i nodi che la compongono e non in una memoria centralizzata. La particolarità di questi modelli è la capacità di adattarsi alle informazioni ricevute in input durante la fase

[26] https://www.massey.ac.nz/~wwpapajl/evolution/assign2/AA/paraproc.html#:~:text=The%20Parallel%20Distributed%20Processing%20(PDP,time%2C%20parallel%20to%20each%20other.

di apprendimento e di modificare la propria struttura (nodi e interconnessioni) in base ad esse, per questo motivo si presentano come sistemi altamente "responsive". Oltre alle enormi abilità e capacità appena descritte, si può affermare che il limite principale di questi strumenti, come già accennato in precedenza, è il loro funzionamento a scatola chiusa (in inglese "Black Box") che dipende dall'impossibilità di analizzare i singoli stadi di elaborazione durante il processo computazionale. Ciò che possiamo osservare delle reti neurali, oltre ai dati in input con cui vengono alimentate durante il periodo di learning, sono i risultati forniti in output che asintoticamente tendono alla migliore approssimazione raggiungibile ma, in generale, risulta ancora un'incognita come questi risultati vengano raggiunti e quali siano i fattori determinanti. A valle dell'analisi svolta in questa tesi, si cercherà di fornire delle metodologie e delle tecniche utili alla spiegazione dei modelli Black Box applicate agli algoritmi di reti neurali utilizzati.

2.1.1 *Architetture principali di Reti Neurali*

Come accennato, la tipologia di una rete neurale dipende dalla sua architettura e dal modo in cui i neuroni sono interconnessi tra di loro. Ogni diversa configurazione sfrutta tecniche di addestramento differenti[27] ed è utilizzata per svolgere attività e risolvere problemi diversi.

Le principali architetture appartenenti alla famiglia delle reti neurali che possiamo trovare in letteratura sono: le reti feedforward, le reti neurali convoluzionali (d'ora in poi *CNN* dall'inglese *Convolutional Neural Network*), le reti neurali ricorrenti (*RNN – Recurrent Neural Network*).

Le reti feedforward sono modelli in cui le informazioni viaggiano e vengono elaborate in modo unidirezionale tra il layer di input e quello di output. Queste reti posso avere diverse strutture in relazione al numero di strati da cui sono composte. Le reti feedforward monolayer hanno una forma estremamente semplice, la loro architettura comprende dei nodi di input (*input layer*) e uno strato di output (*output layer*). L'informazione si propaga in modo aciclico lungo un'unica direzione partendo dal layer di input e terminando in quello di output.

[27] Le tecniche di addestramento di una rete neurale sono le stesse descritte per il Machine Learning e possono essere usate singolarmente o in sinergia tra di loro a seconda della tipologia di rete neurale

Un noto esempio di rete feedforward ad un solo strato è il perceptron (*percettrone*). Questo modello è stato introdotto da F. Rosenblatt nel 1958 ed essendo la prima rete neurale mai implementata presenta una struttura base formata da un solo neurone caratterizzato da una funzione di attivazione lineare a soglia e da ponderazioni che la influenzano. Il perceptron, accettando solo input binari ed essendo in grado di risolvere operazioni logiche di base, trova buona applicazione nella classificazione dei modelli (la cosiddetta *pattern classification*).

Figura 2.1: Rete forward monolayer

Diversamente dalle precedenti, le reti feedforward a più strati (di cui fa parte anche la rete Mutilayer Perceptron, evoluzione del percettrone, che verrà approfondita successivamente essendo questa rete la più utilizzata e studiata e quindi ottimale per una spiegazione più approfondita in merito al funzionamento e alla modalità di apprendimento di una rete neurale) possiedono una serie di layer nascosti (*hidden layers*, già richiamati precedentemente) tra quello d'entrata e quello di uscita. Ogni strato è connesso a quello precedente e a quello successivo e il segnale viaggia in avanti senza connessioni trasversali o cicliche all'interno della rete. La struttura di questa rete permette interazioni tra neuroni a livello globale.

(a) (b)

Figura 1.2: Rete feedforward multilayer semplice (a), Deep feedforward multilayer (b)

Le reti ricorrenti o feedback sono caratterizzate dalla loro architettura ciclica che influenza notevolmente la fase di apprendimento e le successive performance. Le interconnessioni tra layer, tipiche di questo modello in cui le informazioni uscenti da uno strato superiore sono utilizzate come input per un layer inferiore (ricevono una serie di input e ognuno è correlato all'input precedente e a quello successivo), permettono al sistema di creare una memoria e di conservare informazioni temporali. La capacità di ricordare le diverse relazioni che intercorrono tra un input e gli altri, oltre alle informazioni apprese durante il processo di addestramento, rende queste reti particolarmente adatte a svolgere task in cui è di fondamentale importanza l'individuazione di correlazioni tra elementi come nel riconoscimento vocale, nella traduzione e nel riconoscimento della grafia (grazie all'abilità della rete di riconoscere le relazioni tra vocali e consonanti).

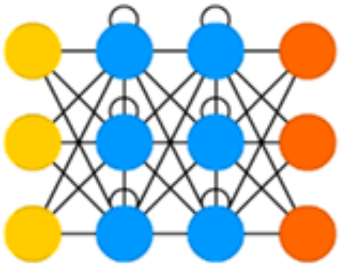


Figura 2.2: Retericorrente (RNN) o feedback

Le Convolutional Neural Network o ConvNet sono anch'esse delle reti feedforward multilayer molto utilizzate per il riconoscimento e la classificazione delle immagini ma per le loro caratteristiche e la loro applicazione meritano di essere trattate separatamente. La struttura e il funzionamento di questi modelli verranno approfondite nei successivi capitoli essendo lo strumento principale utilizzato nell'elaborato e quindi di primaria importanza allo scopo della tesi.

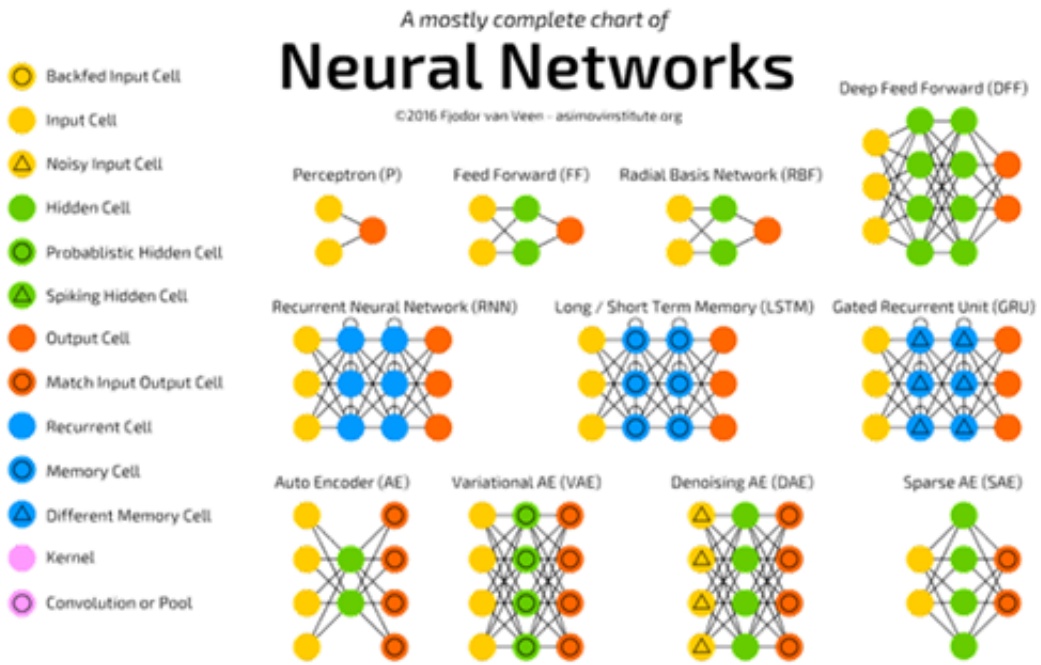


Figura 2.3: vista semi completa delle principali architetture di Reti Neurali citate

2.1.2 *Struttura e funzione di una Rete Neurale*

Le reti neurali artificiali, come illustrato nel paragrafo precedente, hanno strutture e funzionamenti diversi che dipendono dal tipo di architettura del modello e dalla modalità di addestramento di cui si avvalgono per svolgere il proprio compito. Al fine di proporre al lettore una più semplice comprensione generale, utilizzeremo come esempio per questo "drill down" nel mondo delle reti neurali, la Multilayer *Perceptron,* rete molto studiata ed utilizzata dagli "addetti ai lavori" e quindi decisamente rappresentativa e adatta all'intento didattico.

La rete Multilayer Perceptron prende il nome dal *percettrone*, nome del neurone reso famoso da Rosenblatt da cui è composta. Questo modello fa parte della famiglia delle reti feedfoward a più strati, più precisamente è una *deep feedforward neural network* avendo una complessità di struttura e di interconnessioni tra neuroni più elevata di una normale rete feedforward che permette di risolvere dei problemi ingegneristici computazionalmente più importanti. La Multilayer Perceptron è ampiamente utilizzata nella classificazione e per questo motivo si avvale di un apprendimento supervisionato (learning su classificazione di elementi precedentemente etichettati).

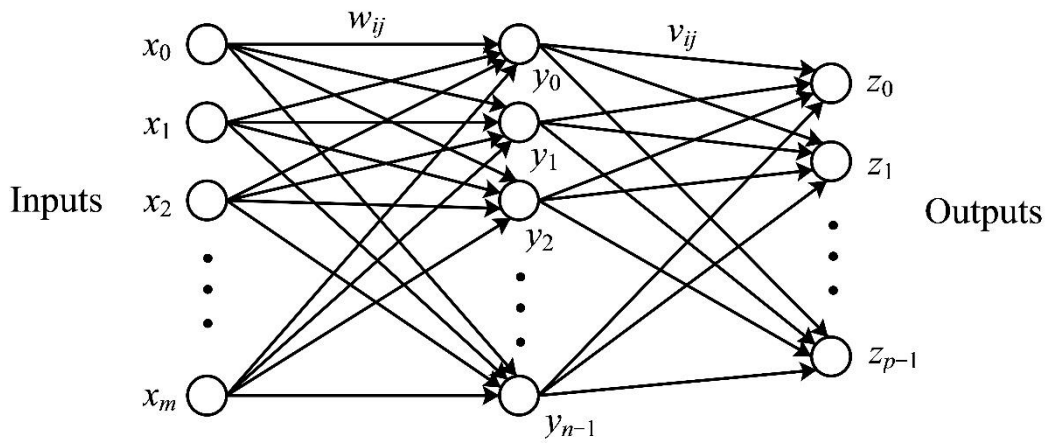


Figura 2.4 tratta da A Quantum Model for Multilayer Perceptron, Shao, C., 2018, arXiv: Quantum Physics: Struttura generica di un Multilayer Perceptron con un hidden layer

Per spiegare come lavora una rete neurale, in Figura 2.5 osserviamo un'architettura generica di un Multilayer Perceptron che utilizzeremo, come già detto, come esempio rappresentativo.

Soffermandoci sul lato di input osserviamo il primo strato (x_0 , x_1, x_2, ... x_m) denominato anche come input layer. Questo strato è composto da un numero *m* di nodi (neuroni) corrispondenti al numero di caratteristiche (*feature*) possedute dai dati in ingresso che andranno ad alimentare la rete. Nel case study di classificazione degli scatti cinematografici proposto in questa tesi, le immagini sono nel formato 16:9 (risoluzione di 160x90 pixel, questo specifico formato e le motivazione per il quale è stato utilizzato verranno approfondite maggiormente nel capitolo riguardante il dataset e nell'appendice) e per questo motivo il primo layer è composto da *14.400 nodi*, uno per ogni pixel dell'immagine.

Scorrendo verso destra nell'architettura delle rete presa in considerazione [Figura 2.5], troviamo gli strati che si posizionano tra l'ingresso della rete e lo strato di output, questi sono chiamati "hidden layer" (già citati in precedenza) e ogni neurone appartenente a questi livelli è interconnesso a tutti i neuroni del layer immediatamente precedente così come a quello immediatamente successivo. Il motivo per cui sono chiamati strati nascosti è per la loro posizione che li rende poco accessibili all'analisi da parte degli studiosi e all'osservazione degli stadi intermedi di elaborazione che avvengono in questi layer (*Black box model*). Le metriche correlate a questi strati intermedi nascosti (numero di layer e numero di nodi

corrispondente ad ogni layer) non sono costanti o fissate a priori ma vengono decise in base alle necessità e all'attività che si vuole che la rete svolga. É buona pratica verificare quale configurazione della rete è in grado di garantire le performance migliori, a valle di questa vengono decisi e definiti il numero di hidden layer e di nodi (di cui sono composti) più adatti.

Il layer finale sulla sinistra della figura (z_0 , z_1, ... z_{p-1}) corrisponde allo strato di uscita della rete (*output layer*). Lo strato di uscita è caratterizzato da un numero di nodi/neuroni che corrisponde al numero di classi in cui i dati devono essere classificati. Ad esempio, se si volesse utilizzare il dataset "Dogs vs Cats[28]" contenente immagini di cani e di gatti per classificare in modo binario le immagini in base all'animale presente, l'output layer sarebbe composto da due nodi, uno per la classe di immagini "cane" e uno per la classe "gatto". In questa tesi, come già spiegato nell'introduzione, l'obiettivo è quello di classificare le immagini nelle otto classi di inquadrature cinematografiche scelte e per tal motivo lo strato di uscita sarà di otto neuroni.

Procedendo nella spiegazione del funzionamento delle reti neurali, vediamo le caratteristiche che descrivono in generale un nodo appartenente ad una rete.

Ogni neurone è caratterizzato da un valore numerico correlato alla scala di colori utilizzata per le immagini in ingresso che, per migliorare il processo a livello computazionale, è ridimensionato in un range tra [0, 1]. Questo valore è denominato "activation" e come si può intuire dal nome, determina l'attivazione del nodo (ad esempio una rete alimentata con immagini in scala di grigi avrà dei nodi in ingresso con valore di attivazione su questa stessa gamma di colori dove il limite inferiore 0 corrisponde al nero e il limite superiore 1 al bianco).

Esattamente come nel cervello umano, ogni attivazione di un neurone (che nelle reti artificiali, così come descritte, corrisponde ad un valore di attivazione poco inferiore o uguale ad 1) influenza l'attivazione dei neuroni dello strato successivo quindi nel momento in cui vengono forniti dei dati in input, per restare in tema pensiamo ad un'immagine, dei neuroni si attiveranno in base alle caratteristiche dell'immagine e questi neuroni metteranno in moto il processo di attivazione dei neuroni dei layer successivi e, man mano che l'informazione viaggia nella rete, altri

[28] Questo dataset è stato creato ed utilizzato per una competizione di Data Analytics lanciata da Kaggle, online community di Data Scientist ed esperti Machine Learning collegata a Google, che consisteva nel trovare un algoritmo di deep learning che classificasse in modo accurato cani e gatti. Maggior informazioni: https://www.kaggle.com/c/dogs-vs-cats

neuroni si attiveranno influenzati dalle attivazioni dei nodi dei livelli precedenti fino a quando il segnale raggiunge lo strato di uscita. I neuroni che compongono l'output layer saranno caratterizzati anch'essi da un diverso valore di attivazione, che per semplicità immaginiamo sempre compreso nello stesso range, e questo valore determinerà la scelta fatta dalla rete neurale in merito alla classe corretta in cui l'immagine deve essere classificata (l'immagine sarà classificata nella classe rappresentata dal neurone con il valore di attivazione più vicino al limite superiore). Entrando più nel merito, iniziamo descrivendo in modo qualitativo come avviene l'attivazione dei nodi in una rete neurale, per poi passare ad una illustrazione di tipo matematico.

Al fine di rendere più chiara la descrizione utilizzeremo come esempio una rete neurale alimentata con le immagini di cifre scritte a mano da 0 a 9 presenti nel dataset MNIST[29], molto in uso tra gli addetti ai lavori.

I numeri contenuti nelle immagini del database MNIST possiedono forme che li definiscono e li caratterizzano (il numero 8 è composto da due cerchi e due linee curve e il numero 7 da due linee), quando una di queste immagini è in ingresso nella rete, layer dopo layer, viene scomposta prima in elementi semplici come i pixel (input layer) per poi giungere, verso l'ultimo hidden layer, ad elementi sempre più complessi e raffinati (cerchi, curve) fino a quando nel livello di output non viene restituita la classe predetta (il numero che è stato individuato dalla rete). Ogni neurone presente negli hidden layer corrisponde ad una forma più o meno complessa in base allo strato a cui si è arrivati, e si attiva nel momento in cui nell'immagine è presente quel determinato elemento grafico. I neuroni che si attivano in un livello influenzano, come già detto, l'attivazione dei neuroni dei layer successivi. Questo avviene perché il nodo del livello successivo corrisponde alla forma complessa e riunificata degli elementi grafici caratterizzanti i neuroni del livello precedente e presenti nell'immagine. Ad esempio, nel caso in cui l'immagine all'interno della rete contenesse un 8 e il primo hidden layer riconoscesse un semicerchio e due linee curve attivando i neuroni corrispondenti a questi elementi grafici, al livello successivo si attiverà il nodo caratterizzato dalla forma che riunisce i tre elementi e che corrisponde al lato superiore di un otto.

[29] Il Database MNIST, estensione del db NIST, è una vasta base di dati di bassa complessità composta di immagini contenuti cifre scritte a mano. Questa collezione di dati è spesso utilizzata per allenare o testare reti neurali che svolgono l'attività di pattern prediction. Maggiori informazioni: https://deepai.org/dataset/mnist

Questo processo di scomposizione e riconoscimento prosegue fino all'output layer in cui, riunificando le forme man mano individuate, verrà riconosciuto un 8 e il neurone corrispondente si attiverà di conseguenza (naturalmente il processo di attivazione/predizione appena spiegato è descrittivo per una rete già addestrata nel riconoscimento di quel determinato tipo di immagini).

Il reale processo di attivazione dei neuroni all'interno di una rete è ancora non del tutto chiaro per gli esperti a causa della grande nebulosità che domina all'interno di queste architetture, si è però comunque provato a fornire al lettore una spiegazione qualitativa in modo da poter affrontare la successiva visione matematica del modello con più elementi possibili ai fini della comprensione.

Il tipo di rete neurale presa in considerazione è composta da livelli definiti "*Fully Connected"*, questo perché al loro interno ogni neurone è interconnesso ad ogni singolo nodo del livello successivo attraverso degli archi caratterizzati da pesi (nella figura 2.5 gli archi corrispondono alle linee che connettono i diversi nodi e il loro peso è rappresentato dai valori: w_{ij} , v_{ij}).

Il valore dei pesi, sempre normalizzato nel range [0, 1], è determinato, in base a diverse ponderazioni e calcoli che affronteremo nel prossimo paragrafo, durante la fase di learning.

Prendendo in considerazione i pesi e valori di attivazione dei neuroni collegati, la misura dell'attivazione di un singolo neurone appartenente ad un hidden layer è data dalla seguente equazione:

$$a = (w_1 a_1 + w_2 a_2 + \cdots + w_n a_n) \qquad (2.1)$$

Al fine di evitare che la complessità matematica cresca esponenzialmente, il valore risultante da questa equazione deve essere linearizzato in un intervallo tra [0, 1] utilizzando una funzione di attivazione. Le più comuni sono la "*Sigmoid function"* e la "*ReLU function*" (unità lineare rettificata).

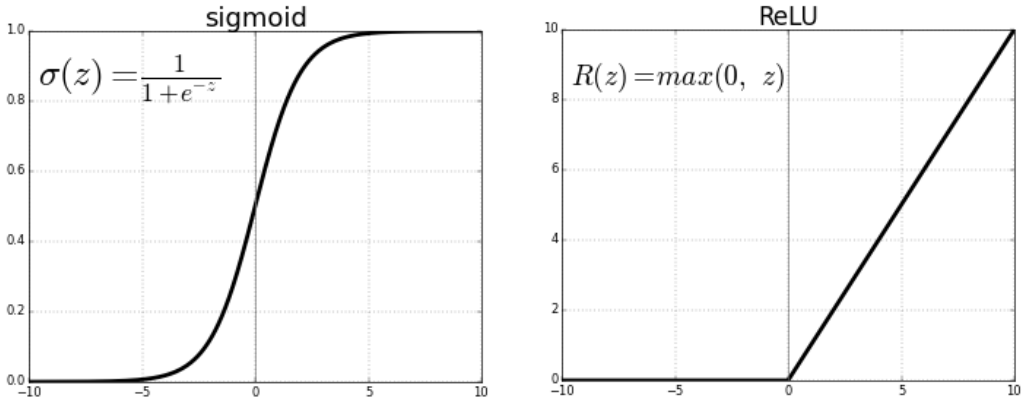


Figura 2.5: Funzione Sigmoid vs ReLU[30]

Utilizzando come esempio per la spiegazione la funzione di attivazione Sigmoid:

$$\sigma = \frac{1}{1+e^{-z}} \tag{2.2}$$

Inseriamo l'attivazione di un singolo neurone al suo interno al fine di linearizzarla, l'equazione precedente diventa:

$$a = \sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n) \tag{2.3}$$

Per completare la formula bisogna introdurre l'ultimo elemento che caratterizza l'attivazione di un neurone, il *bias* o distorsione. Il bias è una soglia che permette al neurone di non attivarsi se questa non viene superata e serve a regolarizzare l'attivazione dello stesso. Aggiungendo il bias all'equazione diventa:

$$a = \sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n + b) \tag{2.4}$$

La formula appena ricavata presenta il valore di attivazione solo per un singolo neurone. Considerando un intero livello la formula ha bisogno dell'aggiunta di indici (vertici e pedici) corrispondenti al neurone e al layer di appartenenza. L'equazione diventa:

[30] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

$$a_0^1 = \sigma\left(w_{0,0}a_0^0 + w_{0,1}a_1^0 + \cdots + w_{0,n}a_n^0 + b_0\right) \quad (2.5)$$

Al fine di rendere la formula più comprensibile proviamo a spiegare gli indici inseriti.

Iniziando dal lato sinistro troviamo a_0^1 che rappresenta la prima attivazione del secondo layer (il primo hidden layer) dove 0, indice inferiore, corrisponde al numero dell'elemento considerato e 1, indice superiore, al layer di riferimento. Spostandoci verso destra lungo l'equazione troviamo i pesi delle interconnessioni e le attivazioni dei neuroni che influenzano l'attivazione del neurone preso in considerazione $w_{0,0}a_0^0 + w_{0,1}a_1^0 + \cdots + w_{0,n}a_n^0$. In questo frammento della formula gli indici di *a* seguono le stesse regole di rappresentazione del valore precedentemente descritto, invece $w_{0,0}$ (così anche gli altri in base a propri indici) corrisponde al peso connesso all'arco che collega il primo neurone del secondo layer e il neurone del layer precedente (il primo pedice rappresenta l'elemento preso in considerazione e appartenente al secondo strato, il secondo l'elemento del precedente layer a cui il neurone è collegato)

Facendo un passo ancora avanti, la formula, in forma vettoriale, corrispondente al valore di attivazione di ogni neurone appartenente ad un livello è la seguente:

$$a^1 = \sigma(W a_0 + b) \quad (2.6)$$

La (2.6) mostra una semplificazione della formula precedente utilizzando una versione vettoriale/matriciale della stessa. In questo scenario W rappresenta una matrice $(m \times n)$ contenete i pesi di tutti gli archi che interconnettono il secondo strato con il precedente (m rappesenta il numero di neuroni del primo layer e n del secondo). I simboli a e b corrispondono invece a vettori con i valori di attivazione (a) e di bias (b) del secondo layer.

2.2 **Apprendimento delle Reti Neurali**

Dopo l'introduzione del funzionamento generale di una rete neurale la domanda che potrebbe sorgere spontanea è; in che modo questi strumenti tanto potenti riescono ad apprendere attraverso i dati forniti e a sfruttare così questo *auto-*

learning per svolgere determinate attività impensabili per un algoritmo informatico/modello matematico standard?

Come già accennato, una rete neurale svolge una fase di addestramento il cui scopo è quello di determinare le metriche illustrate nel paragrafo precedente (pesi e bias). Il valore dei pesi e del bias combinati secondo le equazione descritte, permettono di calcolare i valori di attivazione dei neuroni di ogni layer e quindi rendono la rete in grado di classificare nel modo più corretto i campioni forniti in input (i risultati non sono sempre corretti ma il modello tende asintoticamente alla migliore approssimazione del risultato più accurato).

Le reti neurali possono utilizzare diverse metodologie di learning (Supervised, Unsupervised, Semi-Supervised, Reinforcement). Poiché il metodo più comune per la classificazione di immagini è l'apprendimento supervisionato, ci soffermeremo su questo strumento.

Durante la fase di apprendimento supervisionato, oltre all'immagine, alla rete neurale viene fornita anche la relativa etichetta (in inglese *label)* di classificazione corretta, questa viene utilizzata come mezzo di confronto con la predizione effettuata dal modello sull'immagine campione in merito alla sua classificazione. Questo confronto risulta estremamente importante durante la fase di learning di una rete perché tali strumenti apprendono e migliorano il proprio operato dagli errori commessi e non dalle azioni corrette (semplice intuire la similitudine con il funzionamento della mente umana).

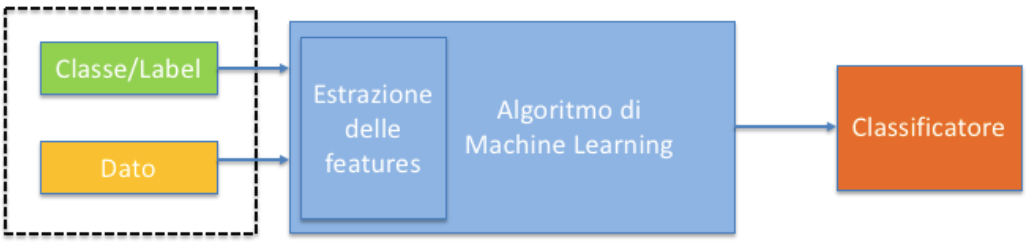


Figura 2.6: Esempio di fase di training supervisionata di una rete neurale artificiale[31]

Andando più nel dettaglio, nel momento in cui la classificazione prevista dal modello coincida con il *label* fornito, la rete lascerà inalterati i propri pesi e il bias supponendo che siano corretti per svolgere la task assegnata. Questa modifica avviene solo nel caso in cui la predizione non sia corretta e il modello, al fine di

[31] https://www.domsoria.com/2018/04/rete-neurale-feed-forward-e-back-propagation/

adattarsi all'errore e migliorare l'apprendimento, effettuerà un ricalcolo di queste metriche così da essere più accurato nella classificazione dei campioni successivi ed evitare di ripetere lo stesso errore.
Utilizzando nuovamente l'esempio di classificazione di immagini di cani e di gatti, se alla rete viene fornita l'immagine di un gatto e la classe predetta dal modello fosse "*cat*", le metriche non verrebbero in alcun modo modificate; in caso contrario, la rete, riconoscendo di aver commesso un errore grazie al confronto tra l'etichetta corretta fornita e la predizione, ricalcolerà i pesi e il bias di conseguenza (tale esempio naturalmente rispecchia anche il caso di classificazione delle diverse inquadrature cinematografiche che si intende studiare in questa tesi).
Il ricalcolo e la ponderazione dei pesi e del bias che avviene campione dopo campione, modificano indirettamente i valori di attivazione dei neuroni dei diversi livelli (un neurone influenza l'attivazione dei neuroni successivi) e permettono alla rete di migliorare il proprio operato ed eseguire una classificazione più accurata.
Durante la fase di learning, questo procedimento di "*adjustment*" delle metriche della rete viene svolto un numero *n* di *epoch*, dove n è il numero di volte che il *training set* (il dataset di immagini ed etichette utilizzato per addestrare il modello) viene processato dalla rete neurale.
Al concludersi di ogni epoch la rete, dopo aver analizzato ogni campione del training set ed aver individuato i pesi più adatti (fino a quell'istante di analisi) a svolgere una corretta predizione delle classi di appartenenza (alla fine di ogni epoch, l'algoritmo calcola un'accuratezza e una loss function delle previsioni del modello in base ai pesi e al bias utilizzati), riceve in input un altro set di dati etichettato da classificare ma che il modello non ha ancora mai processato: questa collezione di dati è chiamata *validation set*.
Il set di validazione ha lo scopo di valutare la bontà della fase di learning svolta fino a quel momento e le prestazioni della rete su dei nuovi dati-Questo processo di classificazione però, diversamente da quello che avviene durante la fase di training, non altera i pesi e il bias del modello ma viene effettuato solo per avere una metrica di confronto utile alla validazione dei risultati ottenuti (alla conclusione di ogni epoch viene calcolato un'accuratezza e una loss function sui dati di validation simile a quella di cui si è fatto cenno prima).
La rete, grazie al processo appena descritto di adattamento agli errori e di modifica delle metriche, anche partendo da un'accuratezza limitata nello svolgere la

classificazione dei campioni (durante i primi step, la classificazione operata dal modello è randomica e utilizzata per inizializzare i parametri ma a seguito di un numero *n* di epoch scelto arbitrariamente in base ai risultati ottenuti dopo diverse prove, migliora notevolmente) riesce a raggiungere asintoticamente la migliore approssimazione di risultato per quanto riguarda l'accuratezza di classificazione dei dati.

2.2.1 *Cost Function, Gradient Descent e algoritmo di Backpropagation*

Un elemento fondamentale da introdurre per analizzare le prestazione del modello in merito allo svolgimento della task ad esso assegnata è la funzione di costo (o funzione di errore) correlata all'algoritmo di training di una rete, solitamente corrispondente alla metrica statistica di somma quadratica degli errori[32] (*SSE – Sum of Squares Error*).

L'obiettivo principale dell'algoritmo di apprendimento di una rete è quello di individuare dei pesi e un bias adatti a minimizzare il valore della funzione di costo, questo perché un costo ingente denota prestazioni non adeguate della rete e quindi un'attività di classificazione dei dati sicuramente non ottimale e accurata.

La funzione di costo è di notevole importanza nel processo di training di una rete perché la rende in grado di autovalutare il proprio operato. Tuttavia, tale funzione è uno strumento "*necessario ma non sufficiente*" per portare a conclusione l'apprendimento del modello costruito.

Per avere una formulazione qualitativa più completa in merito a come le reti neurali apprendono bisogna aggiungere ancora due ingredienti essenziali che rendono il processo estremamente raffinato, sofisticato e matematicamente complesso: la tecnica del Gradient Descent e l'algoritmo di Backpropagation (la spiegazione matematica verrà fornita in un secondo momento).

il metodo del *Gradient Descent* [33] o della discesa del gradiente è un algoritmo iterativo di ottimizzazione in grado di individuare il valore minimo di una funzione a più variabili, permettendo così di avere un modello dai risultati ottimali.

[32] https://365datascience.com/sum-squares/

[33] https://andreaprovino.it/gradient-descent-deep-learning-neural-network/

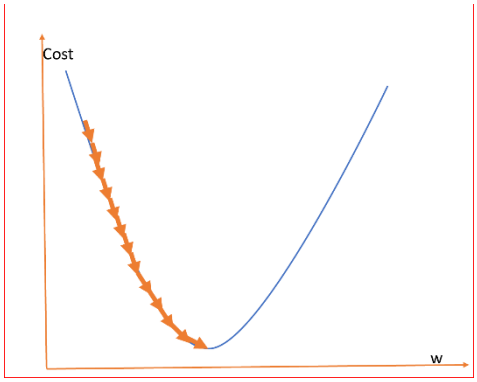


Figura 2.7: Tecnica del Gradient Descent applicata ad una generica funzione di costo

Più dettagliatamente, per comprendere meglio questa tecnica è utile fornire una definizione precisa di gradiente: "*Un gradiente è una funzione a valore vettoriale che rappresenta la pendenza della tangente del grafico della funzione, indicando la direzione della massima velocità di aumento della funzione*[34]".

Come si può intuire, il gradiente fornisce una misura di crescita della funzione, quindi quello che in realtà viene utilizzato per la minimizzazione della funzione è il gradiente negativo.

Nell'implementazione teorica, si parte con la selezione di un punto randomico all'interno dello spazio multidimensionale, di cui si valuta il gradiente e in seguito si individua un secondo punto nella direzione di massima decrescita (valore opposto del gradiente o anti-gradiente) della funzione. Nel caso in cui il gradiente calcolato nel primo punto selezionato è maggiore di quello nel secondo punto è lecito pensare che si stia procedendo nella direzione giusta e si può continuare la discesa fino ad un minimo locale/globale in cui l'algoritmo terminerà; nel caso contrario lo step procedurale verrà diminuito portando l'algoritmo a ripartire in direzione inversa (la differenza tra il raggiungimento di un minimo globale e locale attraverso l'algoritmo di discesa del gradiente verrà fornita in seguito).

Molto efficace appare un esempio cui fanno comunemente riferimento gli addetti ai lavori. Si supponga di essere in cima ad una montagna e lo scopo sia di raggiungere la valle (punto più basso della montagna) senza conoscere il sentiero giusto. La modalità più efficiente per centrare l'obiettivo senza avere informazioni pregresse sul percorso, è quello di procedere iterativamente con dei passi verso la direzione a pendenza più elevata fino a destinazione, raggiunta nel momento in cui

[34] https://lorenzogovoni.com/algoritmo-discesa-del-gradiente/

inizierà la salita (la pendenza tenderà a diminuire fino a diventare nulla e in seguito inversa). Il gradiente applica la stessa metodologia per raggiungere il "punto più basso" della funzione di costo in modo da permettere alla rete di fare previsioni accurate.

Questa tecnica è applicata agli input della funzione di errore (pesi e bias) e sottolinea la rilevanza di ogni peso in merito alle prestazioni di predizione della rete. A tal fine, per migliorare le performance del modello si effettua l'aggiornamento dei pesi (questa modifica dei pesi è stata già accennata nel corso della illustrazione generale della procedura di training) nella direzione negativa del gradiente in modo da minimizzare la perdita.

Il valore di aggiornamento di tali metriche è stabilito dal *learning rate*, misura che influenza il cambiamento dei pesi in funzione del gradiente (il learning rate e la sua funzione verranno meglio analizzati in seguito, durante l'approfondimento matematico del processo).

L'ultimo tassello da inserire per avere una visione completa della fase di training di una rete neurale artificiale è l'algoritmo di Backpropagation.

L'algoritmo di Backpropagation o *Error Backpropagation* [35] può essere considerato come il motore dell'apprendimento delle reti neurali, questo perché effettivamente è la metodologia di correzione dell'errore con cui operativamente si calcola il gradiente e si modificano automaticamente i valori sinaptici (pesi e bias) del modello al fine di minimizzare la funzione di costo e influire positivamente sull'accuratezza delle previsioni.

Il principio di tale tecnica si basa sul confronto/differenza tra il risultato ottenuto (valore di output) e il risultato atteso (obiettivo) che ci fornisce una misura di errore della rete (corrispondente alla functional cost anche se in versione semplificata). Come vedremo nell'approfondimento matematico l'equazione della funzione di costo è leggermente più complessa di una semplice differenza tra output As-Is e output To-Be.

In base a questa misura, l'algoritmo adatta i pesi della rete, aumentandoli o riducendoli, permettendo ai valori di output di confluire progressivamente verso il *desiderata*.

[35] https://www.retineuraliartificiali.net/algoritmo-di-backpropagation/

L'adattamento dei pesi avviene attraverso la propagazione a ritroso dell'errore, utilizzato come base per eseguire l'aggiustamento corretto dei valori sinaptici[36].
L'algoritmo di Backpropagation è suddiviso in due step[37]: Il primo è il *forward pass* in cui il segnale fornito in ingresso al modello stimola l'attivazione dei neuroni appartenenti ai diversi livelli e si propaga verso il layer di output dove viene calcolata l'entità dell'errore con il metodo descritto precedentemente (questo flusso dati all'interno della rete è chiamato *segnale di funzione*). Il secondo è il *backward pass* nel quale l'errore individuato è retro propagato dall'output layer all'indietro (segnale di errore) e i pesi sono aggiornati in modo tale da raggiungere un risultato più accurato all'iterazione successiva (la modifica dei pesi porta ad una conseguente alterazione dei valori di attivazione dei neuroni che compongono la rete, i quali avranno un comportamento differente nella fase di forward pass successiva).

Con la finalità di pervenire ad un'illustrazione il più possibile esaustiva in merito all'apprendimento delle reti neurali e all'applicazione dell'algoritmo di Backpropagation, a seguire proveremo ad approfondire il modello da un punto di vista matematico[38].
Partendo da un neurone i appartenente al livello di uscita, l'equazione corrispondente al segnale di errore durante l'iterazione m (m-simo *sample data* del training set) è definita da:

$$e_i(m) = d_i(m) - y_i(m) \qquad (2.7)$$

In cui $d_i(m)$ rappresenta il risultato atteso in uscita del neurone i e $y_i(m)$ il risultato realmente ottenuto.
L'errore totale calcolato sull'output layer durante l'iterazione *m-esima* è definito con l'equazione:

$$E(m) = \frac{1}{2} \cdot \sum_{i \in C} e_i^2(m) \qquad (2.8)$$

[36] https://www.okpedia.it/algoritmo_back_propagation_reti_neurali

[37] http://www.mtcube.com/SIS/Lezioni5_6_7.pdf

[38] https://www.dsi.unive.it/~srotabul/files/AppuntiRetiNeurali.pdf

nella quale *C* rappresenta il totale dei neuroni che compongono il livello di uscita della rete.

Da questa formula è possibile ricavare l'errore quadratico medio il quale, nel caso in cui *M* corrisponda al totale dei pattern contenuti nel training set, rappresenta la funzione di costo così definita:

$$E_{avg} = \frac{1}{N} \cdot \sum_{j=1}^{N} E(m) \tag{2.9}$$

Come già detto, lo scopo della fase di learning è la minimizzazione della (2.9) attraverso il processo di adjustment dei valori sinaptici liberi (pesi delle interconnessioni e bias) così da permettere prestazioni sempre superiori della rete a livello di accuratezza.

L'operazione di adattamento dei pesi è basata sull'errore misurato per ogni pattern dato in ingresso alla rete. La media di questi aggiustamenti dei valori sinaptici rappresenta una stima della variazione effettiva dei pesi risultante dalle opportune modifiche volte a minimizzare l'equazione (2.9) sull'intero training set.

La tecnica operativa adoperata per raggiungere il minimo della funzione di costo (2.9) è la Gradient Descent.

Il gradiente, in questa sua applicazione, corrisponde all'errore totale dell'output layer (2.8) derivato parzialmente per i pesi delle connessioni dei neuroni dell'output layer con il livello precedente ($\frac{\partial E(m)}{\partial w_{ij}(m)}$).

La variazione di ogni valore sinaptico naturalmente è direzionata verso il gradiente negativo (in senso opposto al gradiente essendo il gradiente una misura di crescita della funzione) seguendo l'equazione:

$$\Delta w_{ij}(m) = -\eta \cdot \frac{\partial E(m)}{\partial w_{ij}(m)} \tag{2.10}$$

In cui η corrisponde al learning rate già introdotto durante la spiegazione qualitativa e utilizzata per influenzare la modifica dei pesi.

Entrando nel dettaglio, per la misura del suddetto gradiente ci si avvarrà di un metodo di calcolo caratteristico delle derivate parziali che ci permetterà di comprendere come ogni parametro influenzi la funzione di costo, ovvero la regola della catena (in inglese nota come *chain rule*) utilizzata come segue:

$$\frac{\partial E(m)}{\partial w_{ij}(m)} = \frac{\partial E(m)}{\partial e_i(m)} \cdot \frac{\partial e_i(m)}{\partial y_i(m)} \cdot \frac{\partial y_i(m)}{\partial v_i(m)} \cdot \frac{\partial v_i(m)}{\partial w_{ij}(m)} \tag{2.11}$$

Come si può notare, attraverso la regola della catena, la derivata parziale dell'errore totale dell'output layer alla m-esima iterazione, per il peso tra il neurone *j* e i è pari al prodotto di quattro derivate parziali.
Nel calcolo di ciascuna derivata prendiamo in considerazione uno dei neuroni di output ($i \in C$) e i risultati sono ottenuti nel seguente modo:

$$\frac{\partial E(m)}{\partial e_i(m)} = \frac{1}{2} \cdot \sum_{k \in C} \frac{\partial e_k^2(m)}{\partial e_i(m)} = e_j(m) \tag{2.12}$$

$$\frac{\partial e_i(m)}{\partial y_i(m)} = \frac{\partial (d_i(m) - y_i(m))}{\partial y_i(m)} = -1 \tag{2.13}$$

$$\frac{\partial y_i(m)}{\partial v_i(m)} = \frac{\partial \varphi[v_i(m)]}{\partial v_i(m)} = \varphi'[v_i(m)] \tag{2.14}$$

$$\frac{\partial v_i(m)}{\partial w_{ij}(m)} = \sum_{k=0}^{m} y_k\,(m) \cdot \frac{\partial w_{ik}}{\partial w_{ij}} = y_j(m) \tag{2.15}$$

Da queste equazioni possiamo ricavare:

$$\frac{\partial E(m)}{\partial w_{ij}(m)} = -\,\delta_i(m) \cdot y_i(m) \tag{2.16}$$

In cui $\delta_i(m)$ rappresenta il *gradiente locale* calcolato nel modo seguente:

$$\delta_i(m) = -\frac{\partial E(m)}{\partial v_i(m)} = -\frac{\partial E(m)}{\partial e_i(m)} \cdot \frac{\partial e_i(m)}{\partial y_i(m)} \cdot \frac{\partial y_i(m)}{\partial v_i(m)} = e_i(m) \cdot \varphi'[v_i(m)] \tag{2.17}$$

Giunti a questo punto siamo in grado di dare una definizione analitica della variazione del peso tra i neuroni *i* e *j* correlato all'm-esimo sample di dati del training set:

$$\Delta w_{ij} = \eta \cdot \delta_i(m) \cdot y_i(m) \tag{2.18}$$

Lo scenario preso in considerazione finora è quello che descrive il procedimento matematico dell'algoritmo di Backpropagation per un neurone di output, per il quale è possibile operare la variazione dei valori sinaptici con lo scopo di minimizzare la funzione di costo attraverso un confronto dell'output reale con il risultato desiderato.

Il comportamento dell'algoritmo per i neuroni appartenenti ad un hidden layer è invece differente perché non si ha a disposizione un valore di output di confronto come nel caso precedente, infatti in questo caso la funzione di costo è definita in modo ricorsivo attraverso il segnale di errore proveniente dai nodi dei livelli circostanti, interconnessi con il neurone preso in considerazione.

L'equazione del gradiente locale, prima definita mediante la (2.17), per un nodo dell'ultimo livello nascosto è la seguente:

$$\delta_i(m) = -\frac{\partial E(m)}{\partial v_i(m)} = -\frac{\partial E(m)}{\partial e_i(m)} \cdot \frac{\partial y_i(m)}{\partial v_i(m)} = e_i(m) \cdot \varphi'[v_i(m)] \qquad (2.19)$$

Della quale:

$$\frac{\partial E(m)}{\partial y_i(m)} = \frac{1}{2} \cdot \sum_{k \in C} \frac{\partial e_k^2(m)}{\partial e_k(m)} \cdot \frac{\partial e_k(m)}{\partial y_k(m)} \cdot \frac{\partial y_k(m)}{\partial v_k(m)} \cdot \frac{\partial v_k(m)}{\partial y_i(m)} = -\sum_{k \in C} e_j(m) \cdot \varphi'[v_k(m)] \cdot w_{ki}(m) \qquad (2.20)$$

in cui, avendo preso in analisi un nodo appartenente ad un hidden layer $i \notin C$ insieme a dei neuroni di output, abbiamo ottenuto che $\delta_i(m) = e_k(m) \cdot \varphi'[v_k(m)]$ da cui ricaviamo la seguente espressione:

$$\frac{\partial E(m)}{\partial e_i(m)} = -\sum_{k \in C} \delta_k\,(m) \cdot w_{ki}(m) \qquad (2.21)$$

Come ipotesi iniziale, abbiamo presupposto che *i* fosse un nodo interconnesso a neuroni appartenenti all'output layer (ultimo hidden layer) ma le formule introdotte hanno validità anche nel caso in cui il nodo *i* fosse collegato ad un altro hidden layer. In tale caso la sommatoria della (2.21) non è estesa ai nodi dell'output layer ma esclusivamente ai neuroni dei livelli successivi. Al fine di poter effettuare questa estensione bisogna definire il peso $w_{ki} = 0$ (ipotizzando i non adiacente a k) e considerare la sommatoria su tutti i neuroni nel modo seguente:

$$\frac{\partial E(m)}{\partial y_i\,(m)} = -\sum_k \delta_k\,(m) \cdot w_{ki}(m) \qquad (2.22)$$

Così il gradiente locale calcolato sul neurone i appartenente all'hidden layer considerato e relativo al m-esimo sample di dati del training set è:

$$\delta_i(m) = \varphi'[v_i(m)] \cdot \sum_k \delta_k\,(m) \cdot w_{ki}(m) \tag{2.23}$$

Provando a generalizzare i due scenari di modifica[39] dei pesi appena descritti, abbiamo che la variazione è fornita da:

$$\Delta w_{ij}(m) = \eta \cdot \delta_i(m) \cdot y_j(m)$$

(2.24)

In cui $\delta_i(m) = \begin{cases} e_i(m) \cdot \varphi'[v_i(m)] & , \ se\ i\ \text{è}\ un\ neurone\ di\ output \\ \varphi'[v_i(m)] \cdot \sum_k \delta_k\,(m) \cdot w_{ki}(m), & altrimenti \end{cases}$

L'algoritmo di Backpropagation descritto è denominato apprendimento *on-line* perché la rete utilizza un sample dati alla volta per il processo di learning ma è presente anche una versione detta *off-line* o *batch* che adopera l'intero training set in fase di apprendimento.
L'apprendimento batch è caratterizzato da una funzione di costo:

$$E = \frac{1}{2} \cdot \sum_n \sum_{i \in C} e_i^2(m) \tag{2.25}$$

E la variazione dei pesi sinaptici:

$$\Delta w_{ij}(m) = \eta \cdot \sum_n \delta_i(m) \cdot y_j(m) \tag{2.26}$$

Queste ultime equazioni sono state introdotte per fornire una panoramica esauriente della fase di training di una rete neurale artificiale ma l'apprendimento off-line è un metodo meno coerente rispetto all'apprendimento del cervello biologico e quindi meno interessante dal punto di vista didattico e sperimentale.

[39] Tutti i passaggi illustrati per il calcolo della variazione dei pesi sono validi anche per l'adjustment del bias, valore sinaptico che influenza anch'esso notevolmente la ricerca della direzione giusta verso il risultato più accurato per una rete

2.2.2 *Learning Rate, Overfitting, Underfitting e metodi di risoluzione*

Nei precedenti paragrafi si è citato più volte il tasso di apprendimento (o *learning rate*[40]), parametro la cui scelta è fissata a monte della fase di training ma che influenza abbondantemente il comportamento dell'algoritmo di Backpropagation e quindi va effettuata con criterio.

La definizione di un learning rate con valore troppo piccolo porterà ad una lenta convergenza al minimo della funzione di costo richiedendo più tempo per la ricerca dei valori sinaptici adeguati al raggiungimento dello scopo. D'altra parte, un fattore di apprendimento elevato potrebbe causare un comportamento altalenante e instabile della rete che potrebbe non raggiungere il minimo globale della funzione.

Una tecnica molto usata dagli "addetti ai lavori" per aumentare il learning rate evitando i problemi descritti è l'inserimento del *momentum* (α numero positivo) all'interno dell'equazione della variazione dei pesi:

$$\Delta w_{ij}(m) = \alpha \cdot \Delta w_{ij}(m-1) \cdot \eta \cdot \sum_n \delta_i(m) \cdot y_j(m) \qquad (2.27)$$

L'utilizzo di questo parametro evita che la funzione raggiunga un minimo locale prima di giungere al minimo globale, obiettivo dell'algoritmo.

Un altro problema riscontrabile in caso il learning rate non venga scelto con attenzione, e in cui la rete potrebbe dirigersi, è la difficoltà di generalizzazione dell'apprendimento su dati non appartenenti al set di training.

Questo problema può portare a due fenomeni importanti, i quali inficiano i risultati della rete addestrata: L'*Overfitting* e l'*Underfitting*.

L'Overfitting[41] o overtraining è un effetto dovuto al sovra-addestramento di una rete che tende a memorizzare i dati forniti durante la fase di training e smarrisce l'abilità di generalizzazione.

Le prestazioni nei modelli affetti da questo fenomeno sono ottime sui dati che compongono il training set ma di gran lunga inferiori se non pessime per quanto

[40] https://www.youtube.com/watch?v=jWT-AX9677k&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=9

[41] https://www.youtube.com/watch?v=DEMmkFC6IGM&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=12

riguarda campioni di dati esterni al set di addestramento come, ad esempio, i dati del validation set.
La valutazione operativa utilizzata per conoscere se un modello è in Overfitting avviene attraverso il confronto tra le metriche di accuratezza di classificazione dei dati del training set e del validation set. In caso si riscontri in fase di learning un notevole gap tra le due metriche, si è di fronte ad una rete sovra-addestrata.
L'Overfitting a livello analitico rappresenta il raggiungimento, da parte dell'algoritmo di Backpropagation, di un minimo locale della funzione di costo invece di uno globale, che porterebbe nettamente a risultati più performanti.
Questo problema deriva, come già accennato, da una scelta di un tasso di apprendimento non corretta perché, in caso la scelta ricada su un valore piccolo del learning rate, questo potrebbe dirigere la rete verso il primo minimo incontrato durante il percorso di minimizzazione che, con molta probabilità, risulterà essere locale. In caso invece il tasso sia troppo elevato si potrebbe incorrere nel rischio di oltrepassare il punto di minimo globale non riuscendo, in conseguenza, a far convergere la funzione di costo.
Esistono diversi metodi per gestire e risolvere il fenomeno dell'Overfitting: uno dei più utilizzati è quello che prevede l'aumento dei campioni all'interno del training set per tutte le classi (non alternando il bilanciamento tra i campioni delle diverse classi) al fine di introdurre più diversità nel set di addestramento che permette alla rete di apprendere con più efficienza riducendo i problemi di adattamento.
Un ulteriore metodo per ridurre il rischio di Overfitting e che agisce sempre sul dataset di partenza è la tecnica chiamata *Data Augmentation*. Tale metodologia consiste nell'aumentare il volume dei dati contenuti nel training set aggiungendo altre immagini derivanti dall'applicazione di alcune trasformazioni ai dati di cui si è già in possesso (zoom, rotazione, ritaglio, filtri, bianco e nero, cambio colore).
La data Augmentation è efficace al fine di gestire il sovra-addestramento perché per una rete neurale, la stessa immagine presentata con trasformazioni diverse, è considerata come due immagini, simili ma separate e quindi permette al modello di avere più campioni per ogni classe su cui effettuare l'apprendimento.
Sono presenti anche altri due metodi per impedire l'adattamento eccessivo della rete ai dati che però non riguardano il dataset di partenza.
Una delle cause di Overfitting potrebbe essere anche l'elevata complessità della rete neurale che stiamo utilizzando. In questo caso il problema si gestisce semplificando

l'architettura del modello (rimozione di layer o riduzione del numero di neuroni per layer).

L'ultima tecnica di gestione del sovra-adattamento della rete è l'aggiunta del "*Dropout*" ad uno o più layer presenti nella rete. Il Dropout applicato ad un livello del modello fa sì che durante la fase di training, un subset (definito in percentuale) casuale di nodi del layer sarà disattivato e non parteciperà al processo di apprendimento. Questa inibizione di alcuni neuroni riduce la complessità della rete in modo casuale e quindi aiuta il modello a generalizzare le lezioni apprese durante la training phase.

Come già accennato, durante la fase di training di una rete si può verificare anche il fenomeno contrario al sovra-adattamento del modello, l'Underfitting.

Il problema dell'addestramento limitato della rete ha origini simili all'Overfitting ma l'effetto e i metodi di risoluzione, come si può immaginare, sono opposti.

Una rete che incorre nell'Underfitting ha performance non adeguate e accurate sia sui dati contenuti nel training set che su quelli di validazione non utilizzati per l'addestramento e una loss function elevata durante ogni tentativo di classificazione.

La gestione di questo fenomeno può essere effettuata con l'aumento della complessità del modello, aggiungendo layer e/o numero di nodi per livello (un modello molto semplice potrebbe avere problemi nel classificare dati con molte caratteristiche) o naturalmente con la riduzione in percentuale del livello di Dropout applicato ad uno o più livelli. Il Dropout non agisce sulla classificazione dei dati del validation set ma solo su quelli di training, quindi in caso si noti che l'accuratezza sui dati di validazione sia ottimale rispetto a quella sul training set, è possibile che tale discrepanza di performance sia dovuta ad un livello di Dropout eccessivo, quindi la diminuzione in percentuale di tale tecnica porterà subito ad un miglioramento dei risultati.

2.3 **Transfer Learning**

Altro argomento di fondamentale importanza, largamente applicato all'interno della sperimentazione di questa tesi riguardante la classificazione delle inquadrature

cinematografiche, e che per tal motivo merita una trattazione separata, è il *transfer learning* (già accennato durante l'introduzione del Deep Learning).

Allenare una rete neurale "*from scratch"* (o da zero) in modo da farle risolvere un determinato problema in modo accurato, rende necessario il reperimento di un dataset di dati di training molto vasto e per giunta correttamente etichettato, operazione spesso difficoltosa.

Inoltre, il complesso processo di addestramento, così come descritto precedentemente, di una *Deep Neural Network* molto estesa (come le reti CNN) su un grande volume di dati, può richiedere un lasso di tempo che va dai giorni a settimane, dipendentemente dalle dimensioni del training set e della rete.

Una soluzione alle problematiche citate è proprio l'utilizzo dell'approccio alternativo di training di una rete neurale chiamato "*transfer learning"*.

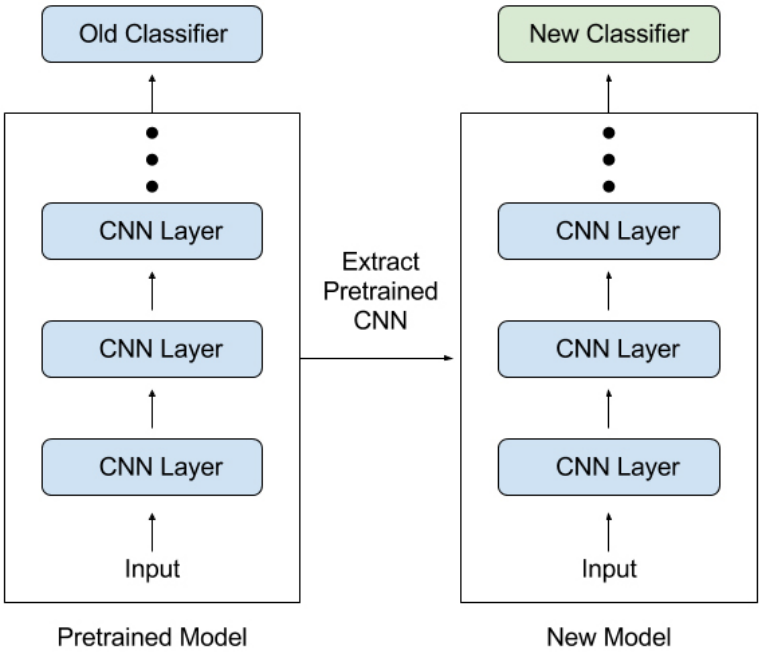


Figura 2.8: Esempio di approccio Transfer Learning per una rete CNN

Il processo di transfer learning permette di evitare il dispendioso allenamento di una rete da zero, riutilizzando alcuni parametri di un modello già pre-trainato su un problema affine all'obiettivo, questo può avvenire in due modalità: il riutilizzo delle features di una rete o il *"fine tuning"*.

2.3.1 *Riutilizzo delle features di una rete neurale*

Questa tecnica utilizza una rete *"pre-trained"* su cui non si effettua nessun "aggiustamento" (come avviene nel fine tuning) e da cui si estraggono dagli hidden layer le feature derivanti dalla fase di addestramento. Queste permettono di effettuare il training di un classificatore esterno (es. Naïve Bayes) in modo che sia in grado di classificare i pattern utili alla risoluzione del problema di interesse.

2.3.2 *Fine Tuning*

Il fine tuning invece consiste nel conservare ciò che una rete ha imparato durante la precedente fase di training. Questo avviene riadoperando i valori sinaptici (pesi e bias) della rete e soffermandosi sul solo addestramento dei livelli finali (uno o più di uno dipendentemente da quanto la task per cui è stata allenata la rete pre-trainata sia similare all'obiettivo prefissato) incaricati della classificazione delle features provenienti dagli strati intermedi, in modo che il modello possa risolvere il problema di interesse.
Operativamente la tecnica di fine tuning di una rete pre-addestrata si effettua sostituendo l'output layer (o anche altri livelli precedenti) con un nuovo livello d'uscita caratterizzato dal numero giusto di classi per risolvere il nuovo problema. I livelli non sostituiti invece vengono resi "*read only"* congelando i valori sinaptici in modo tale che nella successiva fase di addestramento questi siano conservati.
In seguito, viene fornito il nuovo dataset in ingresso alla rete e si procede ad una fase di addestramento che permetterà a questa di adattarsi al nuovo scopo, inserendo nel proprio bagaglio di apprendimento le features del nuovo training set.
Utilizzando un esempio tipico degli addetti ai lavori, immaginiamo di avere una rete addestrata a classificare accuratamente una moltitudine di automobili- noi vorremmo che questa diventi in grado di svolgere una task simile come riconoscere vetture pesanti (es. Camion). La rete pre-addestrata ha già le conoscenze per riconoscere le principali caratteristiche di un camion (fari, ruote, finestrini, specchietti, tergicristalli) perché comuni anche alle auto ma, non avendo mai ricevuto in input un'immagine del genere, potrebbe erroneamente classificare un camion come un tipo di automobile ad esso familiare e presente nel proprio bagaglio

di classificazione. Inoltre, per lo scopo prefissato (la classificazione dei camion) non interessa che la rete abbia le conoscenze per classificare tanti tipi di automobile (conoscenza superflua).

Attraverso il fine tuning, congelando i livelli di partenza e riaddestrando i layer di classificazione su un dataset contenente immagini di camion, è possibile reindirizzare la rete a svolgere questo nuovo compito con grande accuratezza, sbarazzandosi della conoscenza superflua acquisita dal primo training e conservando solo ciò che è utile alla risoluzione del problema di interesse.

Il vantaggio principale derivante dall'utilizzo di questo approccio, oltre al risparmio di tempo nel costruire l'architettura di una rete complessa, è la possibilità di addestrare una rete senza partire da zero ottenendo un processo molto più rapido di uno tradizionale (riduzione del tempo macchina) anche attraverso l'utilizzo di un dataset di piccole dimensioni (diversamente da un addestramento from scratch che richiede un vasto volume di dati) che serva a fornire solamente le feature per lo svolgimento della nuova task ai layer di output. Un altro fattore pro è la possibilità di riutilizzare il comportamento di una rete allenata ad estrarre in modo efficace le caratteristiche fondamentali dai dati in input (solitamente le reti pre-addestrate sono create da esperti del settore e svolgono il loro compito in modo ottimale ottenendo accuracy elevatissime) migliorando notevolmente l'accuratezza rispetto ad una rete neurale custom e limitando sensibilmente l'elaborazione richiesta (numero di parametri da ottimizzare minore).

Con questo ultimo approfondimento si conclude il capitolo secondo dell'elaborato e l'overview generale in merito alle reti neurali artificiali.

Nel successivo capitolo si andrà più nello specifico affrontando il tema delle reti neurali convoluzionali, modello spesso associato dagli esperti alla classificazione delle immagini grazie alla sua abilità nel riconoscimento dei pattern tipici di una figura e ai risultati accurati che esse forniscono e l'ensemble learning, tipologia di apprendimento automatico che, come si intuisce dal nome, permette l'aggregazione di più modelli di deep learning e ha lo scopo di migliorare le performance di classificazione. Queste particolari reti insieme all'apprendimento ensemble, sono estremamente importanti ai fini della tesi perché saranno le tecniche maggiormente utilizzate allo scopo di classificare le nostre inquadrature cinematografiche.

Capitolo 3: RETI NEURALI CONVOLUZIONALI ED ENSEMBLE LEARNING

3.1 Reti convoluzionali: introduzione, architettura e differenze con le ANN

Prima di inoltrarci nel lavoro di classificazione delle inquadrature cinematografiche e di addentrarci nella fase di sperimentazione della tesi, ritengo necessario approfondire uno degli strumenti che più verrà utilizzato nell'arco di tutta l'analisi per raggiungere l'obiettivo dell'elaborato e che è stato brevemente introdotto nei precedenti capitoli: le reti neurali convoluzionali (*Convolutional Neural Networks*).
Questa particolare rete neurale, denominata anche *CNN* o *ConvNet,* abbreviazione del termine di origine anglosassone, è un modello che deve la sua fama e il suo ampio utilizzo nella *computer vision*[42] all'abilità nel riconoscere i pattern e nel classificare dati multidimensionali rappresentabili graficamente e disposti su uno spazio visivo (immagini, video e audio[43]).
Grazie a questa capacità, le reti convoluzionali sono state spesso utilizzate nell'analisi delle immagini e più precisamente nell'identificazione, con una certa accuratezza, di cosa l'immagine rappresenti.
In realtà, tramite una CNN, un computer è in grado di identificare e classificare il contenuto di un'immagine, ovvero gli oggetti presenti al suo interno (ad esempio se un'immagine contiene un cane o un gatto o delle persone).
Questa abilità differisce leggermente dal classificare la tipologia di un'immagine come può essere un'inquadratura cinematografica e quindi un *Campo Lungo* o un

[42] La definizione in letterature di Computer vision è: *"il fenomeno che rende le macchine come computer o telefoni cellulari in grado di vedere l'ambiente come l'uomo"* (https://lorenzogovoni.com/computer-vision/).

La Computer Vision è un'ala dell'intelligenza artificiale e ha il difficile scopo di implementare un computer che riesca a replicare l'intelligenza visiva del cervello, per tal motivo è chiamata anche "*visione artificiale*".

[43] Il linguaggio naturale e le tracce audio sono dati multidimensionali rappresentabili graficamente su un piano visivo, per questo motivo le ConvNet sono spesso utilizzate per il riconoscimento vocale

Primo Piano ma l'obiettivo sperimentale di questa tesi, come anticipato nell'introduzione, è proprio sfruttare la capacità di riconoscimento del contenuto di una figura e le altre abilità di identificazione dei pattern all'interno dei modelli spaziali possedute dalle reti convoluzionale per svolgere un compito più ampio come la classificazione dell'immagine stessa in diverse tipologie (le inquadrature) e non solo degli oggetti contenuti al suo interno.
Le CNN, come anche gli altri modelli di reti neurali, sono costruite per svolgere la classificazione di immagini a seguito di un addestramento su un determinato set di dati contenente le features desiderate (dataset con immagini di cani e di gatti per la loro classificazione o ad esempio per svolgere il riconoscimento facciale si avrà un training set con una serie di volti). Per questo motivo non si otterranno mai dei buoni risultati in caso si provi a far identificare ad una rete convoluzionale oggetti con features non presenti nel training set (è possibile far classificare alla rete immagini con caratteristiche diverse da quelle originalmente usate per l'addestramento attraverso il transfer learning, di cui si è accennato nel precedente capitolo, ma anche con l'utilizzo di questa tecnica è necessario che le features delle immagini da analizzare siano parzialmente simili a quelle su cui è stato effettuato il pre-training della rete e in qualunque caso ri-addestrare uno o più layer finali).

Le reti convoluzionali hanno elementi specifici che le differenziano da una comune rete neurale artificiale (ANN) come le Multilayer Perceptron, utilizzata come base per la spiegazione generale delle reti neurali nel precedente capitolo.
Le reti ANN hanno una scalabilità all'aumentare delle dimensioni delle immagini minore rispetto ad una CNN. In una rete neurale stratificata e fully connected come la MLP, ogni neurone appartenente ad un hidden layer e ha un numero di pesi uguale a $w = hgt * wdth * ch$ (altezza, larghezza e numero di canali del colore). Tale strumento è utile per immagini di piccola dimensione ma, come si può facilmente intuire, un leggero incremento delle dimensioni dei dati in input porterebbe ad una crescita esponenziale del numero di pesi per ogni neurone (e.g. un formato di immagine molto usato per alimentare le reti neurali è 224×224×3, questa dimensione non eccessiva corrisponderebbe a 150.528 pesi per ogni neurone della rete). Questo incremento del numero di pesi per ogni neurone esteso all'intera rete farebbe raggiungere una quantità di variabili sinaptiche difficilmente gestibile dalla rete inoltre le MLP, essendo affette da questa scalabilità poco efficiente, rendono necessaria una trasformazione delle immagini, spesso

quadrate\rettangolari e tridimensionali, in vettori monodimensionali finendo per perdere informazioni aggiuntive utili in fase di training.

Questi problemi sono superati dalle reti neurali convoluzionali le quali hanno i layer e i neuroni ad essi appartenenti, organizzati secondo larghezza, altezza e profondità (volume 3D) e, come vedremo successivamente nella spiegazione architetturale, non completamente connessi e con valori sinaptici condivisi, così da permettere di ridurre i parametri (pesi e bias) in gioco Per tale motivo le CNN sono largamente utilizzate nei lavori di analisi delle immagini. Le differenze tra i due modelli sono ancora più chiare nella figura [3.1].

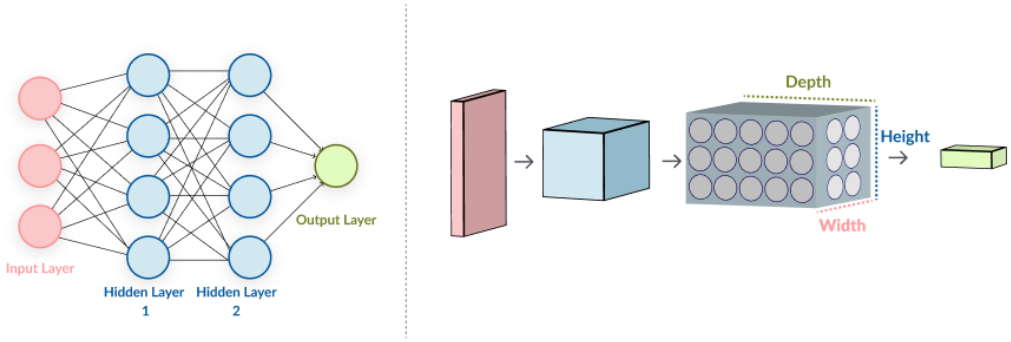


Figura 3.1[44]**:** Architettura di una ANN (sx). Architettura di una rete CNN (dx)

Altro elemento di distinzione tra le CNN e una rete neurale artificiale comune è nella differenza con cui le immagini vengono processate e nell'architettura caratteristica.

Le reti convoluzionali non analizzano l'immagine nella sua interezza ma la scompongono in modelli attraverso filtri di riconoscimento di pattern specifici (più è complessa l'architettura di una CNN, più questa è in grado di identificare schemi complessi). Da un punto di vista strutturale invece le reti CNN sono molto simili alle ANN ma, già detto, diversamente da queste ultime, gli hidden layer non sono tutti completamente connessi (*fully connected*) ma sono presenti anche layer di più tipologie, con neuroni connessi ad una regione ridotta dei livelli precedenti o successivi. Affronteremo più approfonditamente queste ultime differenze, con specifico riferimento all'analisi dell'immagine e alla struttura, nel successivo paragrafo.

[44] https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/

L'architettura di una rete neurale convoluzionale differisce da una comune rete neurale per il suo strato nascosto che non è composto da soli layer fully connected (neuroni interconnessi ad ogni altro neurone dei livelli precedenti e successivi) ma anche da livelli parzialmente connessi.
La struttura tipica di una rete ConvNet è quella che osserviamo nella figura [3.2]

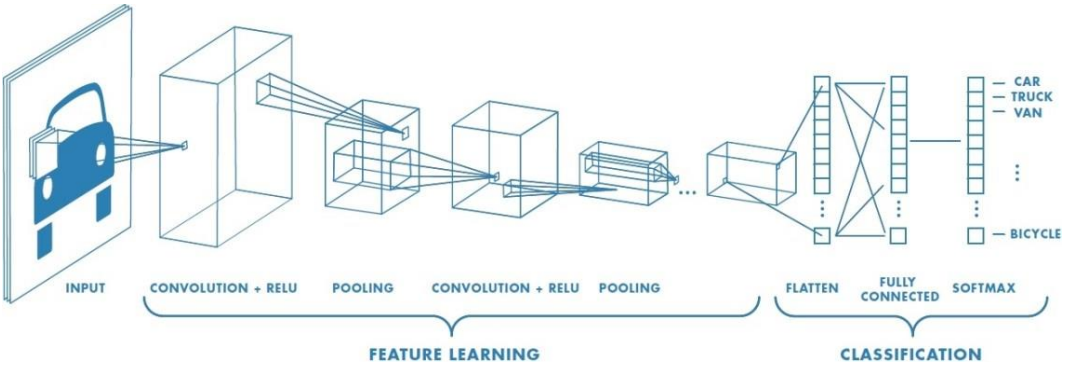


Figura 3.2[45]: Architettura di una rete neurale convoluzionale

L'input layer è lo strato in cui i dati vengono forniti alla rete. Questo livello è strutturato ad hoc sul dato in ingresso secondo le sue features specifiche, ad esempio in caso di un'immagine il livello di input è rappresentato come un insieme di pixel (e.g. h224×w224×ch3).
Il livello convoluzionale è il layer principale da cui prende il nome l'intera rete e che a sua volta è chiamato in questo modo per l'operazione di convoluzione che svolge al suo interno. Lo scopo di questo strato è quello di riconoscere pattern specifici (quadrati, angoli, texture, curvature) in modo efficiente ed accurato. Una CNN può essere formata da più livelli di questo genere e ognuno di questi è in grado di riconoscere un pattern specifico differente.
Si è già detto che aumentando il numero di *Conv Layer* è possibile raggiungere il riconoscimento di caratteristiche sempre più complesse.
Lo strato successivo ad ogni convolutional layer è il *Rectified Linear Units layer* (abbreviato ReLu) che, come la funzione di attivazione lineare da cui prende il nome, ha l'obiettivo di rendere trascurabili i valori negativi derivanti dai layer precedenti.
Altro livello estremamente importante posto sempre dopo il conv layer e il ReLu layer è lo strato di Pool. Questo livello rende in grado di individuare gli elementi

[45] https://www.spindox.it/it/blog/reti-neurali-convoluzionali-il-deep-learning-ispirato-alla-corteccia-visiva/

principali e più caratteristici di un'immagine e di ridurne la dimensionalità eliminando ciò che è superfluo per la rete per svolgere una corretta elaborazione e classificazione (successivamente verranno approfondite le modalità con cui questi livelli operano). L'immagine in output da un livello di pool sarà notevolmente semplificata e più grezza rispetto all'originale.
Infine, nello strato di uscita della rete responsabile della corretta classificazione dei dati in input troviamo livelli FC (Fully Connected) come quelli presenti in una comune ANN. Questo livello interconnette tutti i neuroni dei vari livelli FC in modo da classificare le features identificate nei precedenti livelli secondo una certa probabilità nelle classi predefinite.
Utilizzando come esempio il dataset largamente utilizzato nel machine learning CIFAR-10 [46], abbiamo che un computer può classificare l'immagine analizzata in 10 classi differenti, questa scelta della classe dipende dai risultati raggiunti ai livelli di output dai layer precedenti e quindi dalla classe con maggiore probabilità.
Dopo questa breve introduzione dell'architettura di una CNN vediamo in dettaglio cosa rappresentano i layer principali tra quelli elencati.

3.1.1 *Convolutional Layer*

Come già anticipato, il convolutional layer è il cuore stesso di una CNN, è il livello che rende unica questa tipologia di modello, tuttavia prima di poter descrivere cosa avviene in un layer convoluzionale è opportuno illustrare l'operazione principale che si svolge al suo interno, ovvero la convoluzione.
La convoluzione è il prodotto tra due funzioni di cui una traslata di un valore predeterminato.
Questa operazione, nella sua versione completa e complessa, è notevolmente adoperata nella teoria dei segnali ma nell'ambito della classificazione delle immagini tramite reti neurali e quindi delle inquadrature cinematografiche, la possiamo considerare come un "*filtro"* applicato ad un'immagine.
Il filtro della convoluzione consiste in una matrice (*kernel)* di una dimensione a scelta ($r \times c$) minore di quella dell'immagine su cui è eseguita.

[46] https://www.kaggle.com/c/cifar-10

L'applicazione della matrice sull'immagine altro non è che il prodotto scalare tra il Kernel e il cosiddetto *campo ricettivo*, sottoinsieme dell'immagine di stessa dimensione della matrice (e.g. il campo ricettivo è la sottosezione in alto a sinistra dell'immagine di dimensione 3x3 come la matrice di kernel). Il campo ricettivo su cui il filtro agisce varia a seconda della traslazione di questo sull'immagine. Tale spostamento avviene in base al passo definito, in inglese *stride* (partendo dal lato sinistro verso destra fino al raggiungimento del bordo in cui la traslazione riparte da sinistra scendendo prima, sempre secondo il passo).

L'esito di ogni operazione di prodotto scalare tra la matrice e il campo ricettivo della figura sarà un valore più alto in corrispondenza della caratteristica ricercata con il filtro all'interno dell'immagine.

Alla conclusione dell'intera convoluzione si ottiene un'altra matrice contenente la messa in evidenza di una particolare caratteristica dell'immagine convoluzionata e per questa viene chiamata *feature map*.

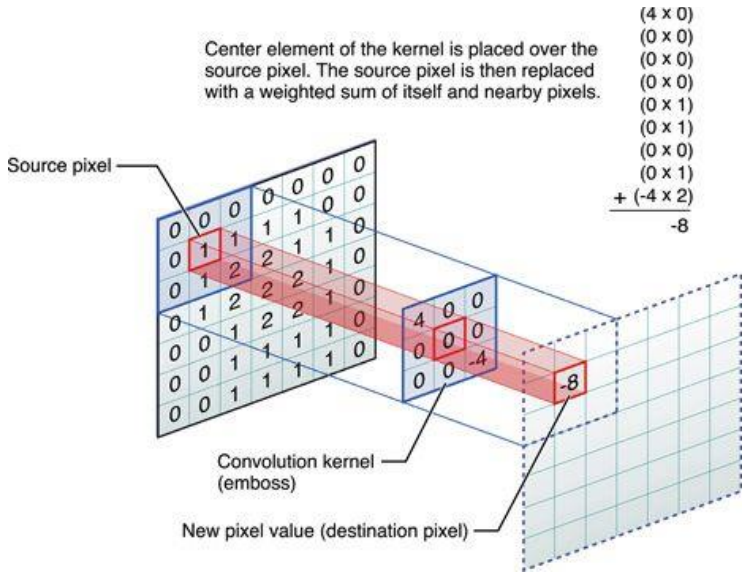


Figura 3.3[47]**:** Immagine originale (a sinistra). Kernel (al centro). Feature Map (a destra)

Con l'aiuto della figura [3.4] è semplice notare come il risultato derivante dall'applicazione del Kernel e le caratteristiche estraibili da un'immagine sono correlate ai valori numerici definiti all'interno della matrice.

[47] https://tech.everyeye.it/articoli/speciale-cos-e-convolutional-deep-learning-cosa-serve-33066.html

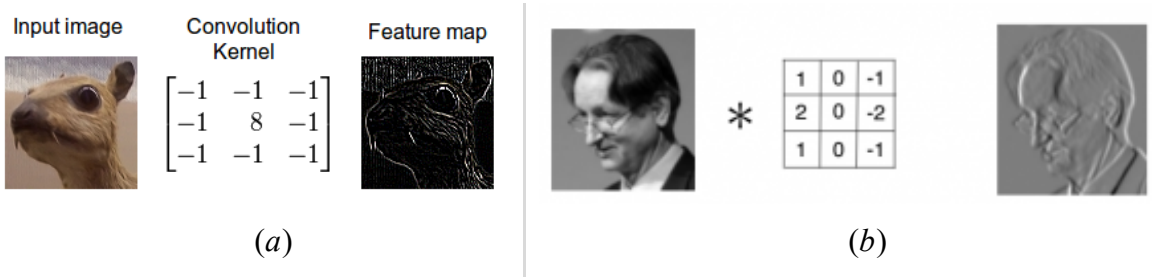


Figura 3.4: Esempio di filtro che cattura i contorni di un'immagine (*a*).
Esempio filtro effetto embossed (*b*)

Mediante l'operazione di convoluzione e quindi l'utilizzo dei kernel è possibile ottenere molteplici filtri applicabili alle immagini, ognuno dei quali, capace di estrarre e porre il focus su features differenti come le sole curve o i soli contorni (e.g. il convolutional kernel in *a* mette in evidenza i contorni di un'immagine mentre quello in *b* crea un effetto embossed).

Tornando a ciò che caratterizza la matrice di kernel, cuore della convoluzione, è importante sapere che ogni filtro coinvolge un numero di valori sinaptici (pesi) pari agli elementi che compongono il kernel (e un bias unico per ogni matrice).

Il numero di parametri in gioco quindi non dipende dalla grandezza dell'immagine (come per i layer delle comuni artificial neural network) che può essere anche di grande dimensioni. Il numero di valori sinaptici per un layer convoluzionale è calcolato secondo l'equazione:

$$\left(\left(\left(kernel_{wth} * kernel_{hgt}\right) * n_{chn}\right) + 1\right)$$

nella quale $kernel_{wth}$ e $kernel_{hgt}$ sono le misure di dimensionamento della matrice e n_{chn} corrisponde al numero di canali (uguale a 3 per le immagini RGB a colori e uguale a 1 se greyscale) per cui ogni kernel è ripetuto. Il più uno rappresenta il bias, comune per ogni matrice.

È questa la caratteristica principale che permette alle CNN di risolvere il problema di scalabilità su immagini di grandi dimensioni o di alta qualità e che le rende particolarmente efficienti nell'analisi delle immagini stesse.

Un livello convoluzionale è caratterizzato da quattro parametri fondamentali, detti anche *hyperparameter*, che influenzano il suo comportamento.

Il primo è la dimensione del kernel (*Kernel size*) corrispondente al numero di righe e colonne della matrice di filtro (e.g. 4x4)

Il secondo è il riempimento zero (*zero-padding*). Questo valore costituisce le dimensioni del bordo esterno posto al volume di input iniziale ed è adoperato al fine di evitare la perdita di informazione tra i diversi layer (i bordi di un'immagine attraverso diverse convoluzioni tendono a sparire).
Un iperparametro già introdotto precedentemente è lo *stride* che corrisponde al passo, misurato in pixel, in base al quale il kernel si sposta sull'immagine individuando nuovi campi ricettivi (velocità di traslazione).
Infine, l'ultimo valore che influenza un livello convoluzionale è il numero di filtri. Un layer che effettua la convoluzione può possedere un numero n di kernel paralleli e applicati alla medesima immagine fornita alla rete così da generare filtri differenti. Ogni filtro può catturare una caratteristica dell'immagine alla volta e ogni convolutional layer è contraddistinto da uno o più filtri di stesso genere e per questo motivo, in una CNN, si inseriscono interi blocchi di livelli che effettuano la convoluzione, così da evidenziare più features possibili utili alla rete per analizzare l'immagine.
Nei primi livelli la matrice cattura caratteristiche "di basso livello" come semplici linee o angolature che man mano andranno a combinarsi nei successivi conv layer, evidenziando forme sempre più raffinate che rappresentano oggetti complessi.
A seguire, i livelli convoluzionali finali avranno filtri di alto livello capaci di identificare strutture articolate come un occhio, un viso, una persona.
La scelta di questi hyperparameter non ha una metodologia standard ma è strettamente correlato ai dati in input.

3.1.2 *ReLU Layer*

L'output di un livello convoluzionale a sua volta diventa input per il layer successivo. Collocato di seguito al conv layer troviamo il livello ReLU.
L'obiettivo di questo strato, come già accennato, è quello di annullare tutti i valori negativi e rendere non lineare l'output derivante da un calcolo lineare come la convoluzione, o per la precisione, il prodotto scalare.
La funzione che interessa questo livello è definita come $f(x) = \max(0, x)$ ed è eseguita su tutti i valori generati dai layer convoluzionali.

Questa operazione porta vantaggi notevoli in termini di rapidità di allenamento di una rete (carico computazionale ridotto eliminando valori negativi), non influenza i campi ricettivi del precedente layer e non influisce in modo percepibile sull'accuratezza.

3.1.3 *Pooling Layer*

Successivamente ai due layer appena descritti, gli "addetti ai lavori" di solito inseriscono un livello che svolge l'operazione di pooling (questo layer può essere collocato anche dopo un intero blocco di livelli convoluzionali e ReLU).
L'output generato dai filtri convoluzionali, le feature map, permette di ottenere informazioni sull'immagine molto più chiare e consistenti rispetto alle originali, questo grazie all'estrazione di alcune caratteristiche ben precise e all'eliminazione del rumore (caratteristiche non di interesse e quindi non messe in evidenza).
Per questo motivo è possibile e necessario ridurre il carico computazionale e rendere più semplice l'elaborazione limitando il peso delle immagini di grandi dimensioni.
Il livello di pooling ha come obiettivo proprio quello di ridimensionare il volume delle immagini conservando però le features principali di questa.
Esistono diverse tipologie di operazione di pooling eseguibili in un pooling layer, la più utilizzata in ambito di reti neurali è quella di *max pooling*.
Il meccanismo di funzionamento del max pooling e quindi del livello di pooling è tanto semplice quanto essenziale per le prestazioni di una CNN.
All'interno del layer viene processata l'immagine, fornita sotto forma di matrice r×c essendo risultato di una convoluzione, attraverso un filtro \matrice di pooling, usualmente di dimensione 2×2, che si muove su di essa secondo un passo della medesima lunghezza.
Il filtro di pooling, attraverso questa traslazione, individua i campi ricettivi (subsample) dell'immagine e di questi trova il valore massimo per ognuno, questo massimo è salvato in una terza matrice di output. Questa operazione è chiaramente visibile nella figura [3.5].

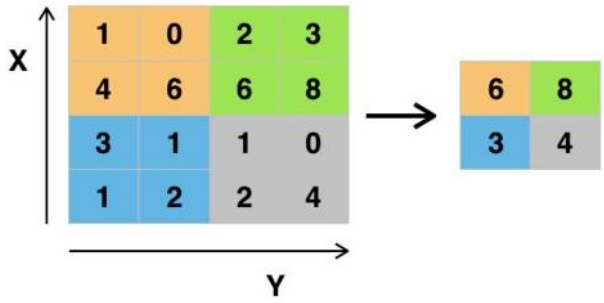


Figura 3.5: Esempio di max pooling. Sulla sinistra l'immagine con i campi ricettivi evidenziati di colore diverso. Sulla destra la matrice di output composta dai max value dei campi ricettivi

Al di là del max pooling, come si è già accennato, esiste anche un'altra tipologia molto adoperata e che conduce allo stesso risultato di riduzione del carico computazionale e miglioramento dell'elaborazione, l'*average pooling*.
Questa tipologia di pool effettua esattamente le stesse operazioni di quella appena descritta ma invece di estrarre il massimo in ogni campo ricettivo, trova la media dei suoi elementi.
Il risultato generato da un livello di pooling non è un banale ridimensionamento dell'immagine, anche se la diminuzione della dimensione spaziale del volume è notevole, perché nonostante questa riduzione dell'informazione, viene conservata la massima/media di ogni sub matrix dell'immagine (le sue features principali) garantendo anche una limitazione dei requisiti computazionali per i successivi layer.
Gli Hyperparameter di un layer di pooling sono i medesimi di uno convoluzionale soprattutto per quanto riguarda le dimensioni del pool, lo zero-padding e il passo.

3.1.4 *Fully Connected Layer*

Posti alla fine della rete neurale convoluzionale, gli FC layer sono incaricati della classificazione delle immagini, quindi della generazione dell'output di una CNN.
Questo strato di livelli completamente connessi riceve in input l'immagine manipolata dai precedenti layer e produce un vettore di N dimensione dove N corrisponde al numero di classi in cui si vuole classificare il volume in input.
Ad esempio, nel caso in esame, le inquadrature cinematografiche sono di otto tipologie quindi il numero N sarà uguale ad 8 poiché questo è il numero di classi tra cui il programma dovrà scegliere (da 0 a 7).

All'interno del vettore N-dimensionale sono presenti le probabilità che l'immagine in input corrisponda a quelle determinate classi.

La metodologia secondo cui il livello FC agisce e affilia le probabilità alle classi è semplice. Il layer riceve in input le mappe di attivazione (derivanti dal ReLU layer) che mettono in evidenza le caratteristiche di alto livello (e.g. occhi, naso, volti...) e identifica quali feature sono in maggior misura collegate ad una classe ben precisa. Utilizzando sempre un esempio proveniente dal caso in esame, se il computer presume che l'immagine sia un primo piano, saranno presenti valori di entità elevata per le mappe di attivazione di feature tipiche dei primi piani, volto, capelli, bocca e così via (lo stesso vale anche per un campo lungo in cui si avranno alti valori per caratteristiche come colline, palazzi...).

Analizzate queste correlazioni, il livello completamente connesso calcola la probabilità relativa ad ogni classe attraverso un prodotto tra i pesi.

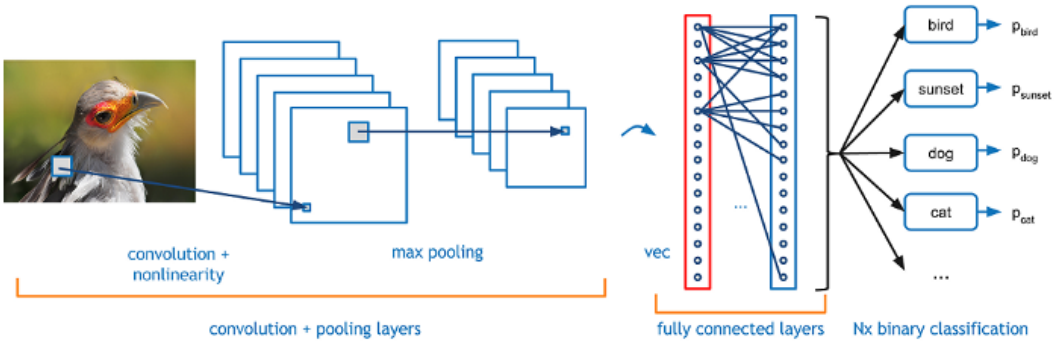


Figura 3.6[48]: Esempio completo del funzionamento di una CNN fino al calcolo delle probabilità per ogni classe eseguito negli FC layer

Riassumendo, le reti neurali convoluzionali sono dei modelli con caratteristiche particolari notevolmente utilizzati nell'ambito della computer vision e più precisamente della classificazione delle immagini grazie alla capacità di riconoscere i pattern e all'architettura che permette a queste di avere un'ottima scalabilità all'aumentare della dimensione dei dati in input.

Come già descritto in dettaglio, le CNN sono composte da una serie di strati che vanno da quello di input, luogo in cui i dati vengono forniti alla rete, seguito da dei blocchi di convolutional, ReLU e pooling layer e in conclusione una sequenza di

[48] https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

livelli fully connected, simili a quelli di una più comune MLP, che operano la classificazione dei dati.

In questa architettura, i layer convoluzionali generano una serie di feature map (insieme di caratteristiche ricercate), successivamente questo output viene processato dal layer ReLU, che elimina i valori nulli e introduce della non linearità nel sistema, e dal livello di pooling che riduce il carico computazionale ed evidenzia le informazioni principali dei dati.

Questo processo avviene in modalità *waterfall* un numero di volte pari al numero di blocchi convoluzionali[49] di cui la rete è dotata (ogni blocco è utilizzato per esaltare alcune caratteristiche dei dati utili per permettere una facile classificazione e oscurare quelle di scarso interesse, le feature evidenziate dipendono dal tipo di analisi che si sta effettuato e sono differenti per ogni problema da risolvere) fino a giungere ai livelli di classificazione standard (e.g. in python vengono utilizzati il flatten layer per rendere l'output monodimensionale e i dense layer per eseguire la classificazione).

L'architettura appena descritta permette alle CNN di filtrare gli input ed estrarre solo le informazioni di maggiore rilevanza per la classificazione.

I parametri in gioco durante il processo non necessitano di un impostazione manuale ma sono appresi dalla rete in modo automatico attraverso piccoli adattamenti e *adjustment* che permettono di migliorare l'accuratezza step by step.

Per concludere, utilizziamo un *use case* esemplificativo, attinente allo scopo dell'elaborato.

Anticipando il tema della classificazione delle inquadrature cinematografiche e immaginando di utilizzare una semplice CNN per svolgere l'operazione, in caso si voglia classificare un *primo piano*) si parte dall'immagine, insieme di pixel molto probabilmente raffigurante il volto di un soggetto, fornita in input alla rete.

L'immagine in seguito giungerà ai primi livelli convoluzionali che estrarranno le sue caratteristiche elementari (spigoli, linee, contorni, curve, colori, nitidezza, luminosità...).

Risalendo lungo la catena architetturale della rete, i successivi layer convoluzionali, utilizzando le informazioni grezze dei livelli precedenti, metteranno in evidenza le caratteristiche di alto livello, permettendo di individuare feature come il naso, la bocca o gli occhi del soggetto in primo piano.

[49] Per blocchi convoluzionali si intende l'insieme di livelli convoluzionali, ReLU e di pooling.

L'astrazione delle informazioni processate ad ogni layer, arrivati ai livelli conclusivi, consente di distinguere un viso dagli altri elementi della figura e quindi di discriminare tra un primo piano e una tipologia come il *campo medio* in cui spesso la figura umana non è presente o è in secondo piano. Tutto questo in modo automatico.

Le CNN di ultima generazione adoperano anche gli *inception modules* i quali operano la convoluzione attraverso la scomposizione del Kernel in sezioni 1x1 in modo da poterla eseguire gradualmente ed efficientare la *memory consume* della GPU (e.g. l'applicazione di un Kernel 4x4 è rimodulata nell'utilizzo di 32 Kernel 1x1 così da rendere il processo più snello facendo eseguire una convoluzione alla volta alla rete e non caricando *in memory* un intero blocco 4x4).

Conclusa l'illustrazione in merito al funzionamento delle reti neurali convoluzionali, nel successivo paragrafo introdurremo a livello teorico una metodologia innovativa, attualmente molto in voga in ambito di apprendimento automatico che verrà poi approfondita e calata nel contesto di analisi nei successivi capitoli, l'Ensemble Learning.

Questo strumento sarà, insieme alle CNN implementate con la tecnica del transfer learning, il modello utilizzato per la classificazione delle tipologie di immagini di interesse (le inquadrature cinematografiche) e quindi il fulcro dell'analisi sperimentale svolta nella tesi.

3.2 **Ensemble Learning**

L'ensemble learning[50] è una tecnica di apprendimento automatico che si basa sulla elaborazione di più ipotesi (collezione), È un sistema complesso nel quale la macchina, partendo da un insieme di dati di addestramento (training set), costruisce un insieme di ipotesi (decision tree) e poi consulta tutti gli alberi decisionali che ha estrapolato. Questi, con una probabilità di errore differente p_e^i dove i indica il relativo albero decisionale, possono fornire anche risposte diverse perché hanno

[50] www.andreaminini.com/ai/machine-learning/ensemble-learning

metodi di ragionamento differenti, tuttavia è probabile che alcuni di essi restituiscano la stessa risposta. A questo punto il calcolatore sceglie la risposta data dalla maggioranza dei decision tree.

Tale sistema utilizza diversi classificatori i quali, collaborando fra loro mediante specifiche operazioni, consentono di ottimizzare la prestazione poiché vengono sfruttati i punti di forza di ogni modello e ne vengono limitate le debolezze [Fig. 3.7]. Più semplicemente possiamo affermare che per ottenere prestazioni predittive affinate vengono aggregati diversi modelli semplici così da ottenere un modello più complesso, chiamato *ensemble model*, che ha una performance superiore (e.g. differenti modelli di reti neurali o come nel caso in esame, lo stesso modello addestrato su dataset di immagini diverse che permettono di migliorare l'accuratezza).

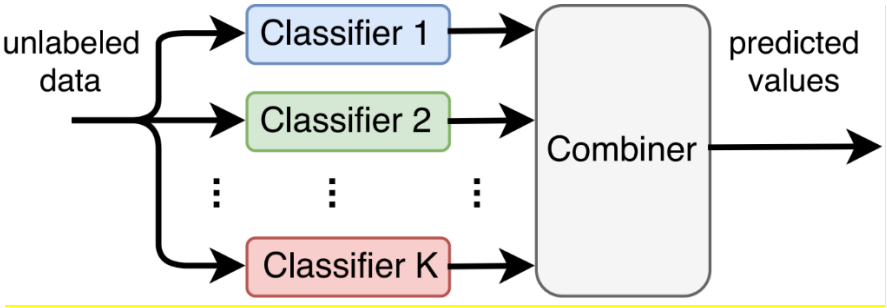


Figura 3.7[51]**:** Struttura generale di un *ensemble model*

Con questa tecnica si limitano le probabilità di errore anche se, laddove si utilizzi un unico training set, il sistema potrebbe presentare una scarsa affidabilità, Ciò si verifica, ad esempio, quando nell'insieme di addestramento vi siano *outliers*[52]. In tal caso, poiché i decision tree, in presenza di un solo training set, non sono del tutto indipendenti fra loro, può accadere che tutti gli alberi decisionali vengano influenzati dal problema e che, in conseguenza, non prendano nella dovuta considerazione situazioni o circostanze importanti. Per superare questo limite conviene usare training set diversi in modo da rendere i vari decision tree davvero indipendenti gli uni dagli altri (per tal motivo, nella fase analitica di questo elaborato saranno costruiti tre training set contenenti immagini differenti, adoperati anche per il fine tuning di tre reti neurali convoluzionali).

[51] https://it.mathworks.com/matlabcentral/fileexchange/68383-ensemble-learning-toolbox

[52] Dato fuorviante fuori dal range di risultato possibile

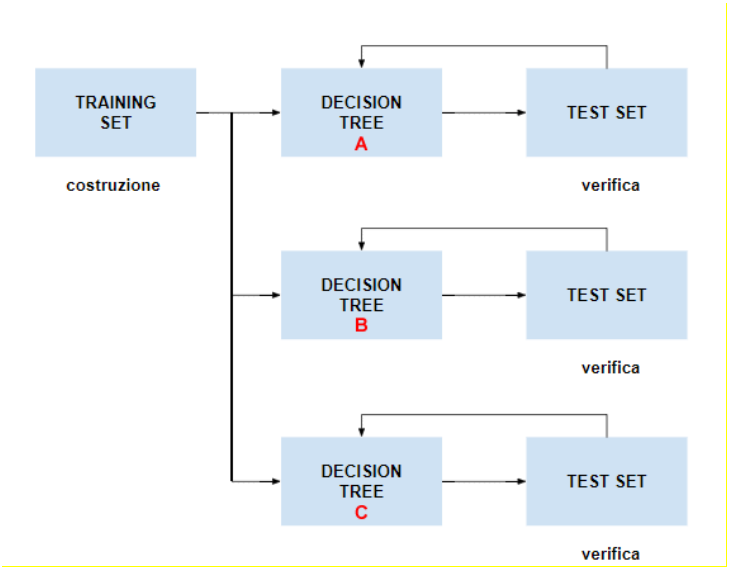


Figura 3.8[53]**:** Costruzione di un albero decisionale

Possiamo dunque affermare che per ottenere un ensemble efficace è utile produrre alberi il più possibile diversi tra loro, affinché l'addestramento di uno non influisca su quello degli altri.

L'Ensemble Learning utilizza classificatori deboli (*weak classifier*)[54] che, sommati in un certo modo tra loro, danno vita ad un classificatore forte (*strong classifier*) il quale raggiungerà un trade-off tra varianza e distorsione ottimale (il modello sarà abbastanza flessibile per svolgere il proprio compito in modo efficiente, gestendo la complessità dei dati in input, senza però aumentare pericolosamente la variabilità e compromettere quindi l'accuratezza). Inoltre, un ensemble model finale che adopera una sola tipologia di weak classifier è detto omogeneo, altrimenti corrisponde ad un modello eterogeneo. Con questa metodologia si contiene il rischio di Overfitting che si verifica quando l'algoritmo si adatta fin troppo bene ai dati di addestramento ma perde in capacità di generalizzazione. Lo scopo da raggiungere è invece quello di consentire al modello la memorizzazione dei dati che ha visto durante l'apprendimento e di essere in grado di generalizzare gli esempi che non ha visto.

L'unione di modelli deboli combina modelli più accurati di una scelta casuale e mira a costruire un modello finale che abbia un alto valore predittivo.

[53] www.andreaminini.com

[54] un classificatore debole riesce a classificare almeno il 50% +1 dei campioni di un problema binario https://lorenzogovoni.com/overfitting-e-underfitting-machine-learning/

Si contano diversi tipi di classificazione ensemble, a seconda delle modalità con cui i dataset vengono costruiti e di come vengono manipolate le features e viene calcolato il risultato finale.
Di regola i classificatori vengono combinati insieme attraverso una media pesata. Questa si basa su un periodo di addestramento dei modelli che potremmo chiamare periodo di prova, con l'obiettivo di determinare il peso di ogni modello nella visione finale. Quindi si giunge ad una media pesata di tutti i modelli utilizzati, producendo la previsione di multimodel ensemble.
Le tecniche di ensemble learning sono varie. Tra queste si richiamano il *bagging*, il *boosting* e lo *stacking*.[55]

3.2.1 *Bagging*

Nel bagging (bootstrap aggregation), modelli dello stesso tipo vengono addestrati su dataset diversi, ognuno ottenuto dal campionamento iniziale. Vengono infatti generate versioni multiple di un classificatore per combinarle in un insieme di classificatori al fine di ridurre la varianza generale ereditata dal dataset di addestramento. Formalmente viene ripetutamente campionato il training set di input, con una distribuzione di probabilità uniforme e con reinserimento al fine di realizzare, ad ogni iterazione, un modello sul campione appena prodotto. Questo procedimento di *boostraping* e le proprietà di approssimazione dei campioni che ne derivano (indipendenza e rappresentatività), permettono di avere modelli quasi del tutto indipendenti nonostante non vengano forniti in input dataset completamente distinti, operazione che richiederebbe una grandissima mole di dati e di *effort* manuale. Come già accennato, l'obiettivo del bagging è la riduzione della variabilità del classificatore finale rispetto ai singoli modelli che, essendo statisticamente distribuiti tutti nel medesimo modo e indicativamente indipendenti, combinati tra loro non modificano la risposta finale ma limitano notevolmente la varianza generale. I classificatori utilizzati nella tecnica di bagging hanno tutti la stessa importanza e, dopo che ogni modello avrà votato l'esito della predizione, la classe che avrà ottenuto la maggioranza dei voti sarà considerato l'output

[55] https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

complessivo (questa metodologia di aggregazione dei risultati viene anche chiamata *hard voting* ed è utilizzata nei problemi di classificazione). Ne è un esempio il *random forest*, costituito da molti algoritmi di alberi decisionali profondi, allenati su campionamenti indipendenti e ognuno caratterizzato da una parte delle feature dei dati selezionata casualmente (ciò permette la non correlazione degli alberi che ricevono in input dati con caratteristiche differenti e forniscono output robusti). Altro metodo di combinazione bagging dei weak classifier per svolgere la classificazione è il *soft voting*, in cui viene effettuata la media delle probabilità generate dai modelli. La risposta scelta sarà quella con la migliore probabilità media. Diversamente, per problemi che non riguardano la classificazione e quindi la scelta di una classe di appartenenza come, ad esempio la regressione, gli output sono combinati attraverso il calcolo della media dei loro valori intrinsechi (e.g. per le reti neurali questi valori sono pesi e bias).

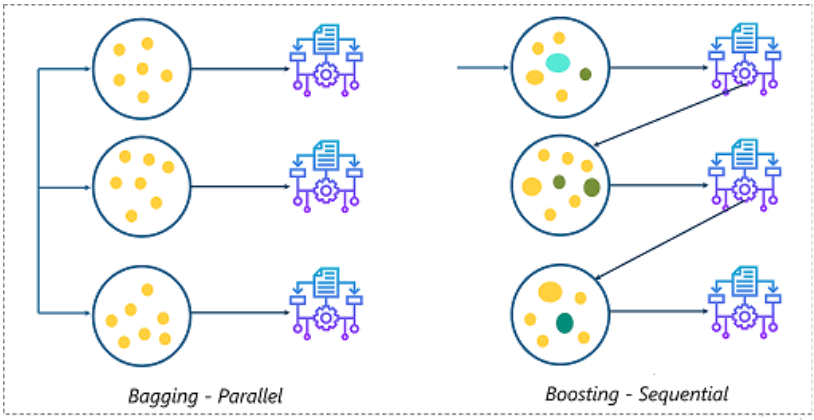


Figura 3.9[56]: Differenza tra la tecnica di Bagging e il Boosting

3.2.2 *Boosting*

Il boosting, diversamente dalla tecnica appena illustrata, persegue l'obiettivo di creare un modello che riduca la distorsione prodotta dai weak classifier (modelli con una varianza limitata ma un bias elevato). Tale metodologia si basa su modelli allenati iterativamente in step diversi dipendenti tra loro e posti in ordine sequenziale. Approfondendo, il boosting attribuisce a ciascun classificatore un certo peso che influirà sulla scelta finale. Il diverso peso dei classificatori viene calcolato

[56] https://prog.world/xgboost-distributed-learning-and-concurrent-forecasting-with-apache-spark/

in base all'errore di accuratezza che ciascun modello commette in fase di learning, quindi il peso aumenta in misura direttamente proporzionale all'avvicinamento alla precisione (l'attenzione dei classificatori è incentrata sui campioni più ostici da lavorare per cui la classificazione ha avuto meno successo, così da migliorare e ridurre il bias). Tale tecnica si divide in due tipologie- *Adaptive Boosting* e *Gradient Boosting*-che si differenziano in base al metodo di ottimizzazione utilizzato per trovare il modello più accurato e con minore distorsione. In via generale, entrambi adoperano la somma pesata dei valori dei weak classifier.

3.2.3 *Stacking*

Il Bagging e il Boosting utilizzano modelli omogenei mentre la tecnica di *Stacking learning* (chiamata anche *stacked generalization*) usa modelli eterogenei (e.g. modelli differenti o lo stesso modello addestrato su dataset molto diversi). Inoltre, questa tecnica non prevede, come il Bagging, che il risultato di output derivi da una votazione ma combina le previsioni di singoli algoritmi servendosi di un ulteriore classificatore detto *meta-classificatore* (e.g. una semplice rete neurale multilayer perceptron, una CNN o un algoritmo SVM[57]).
Entrando più nel dettaglio della spiegazione teorica di questa tecnica che, come vedremo nella fase di sperimentazione sarà lo strumento utilizzato per avere dei risultati accurati durante classificazione delle inquadrature cinematografiche, va evidenziato che lo stacking ensemble learning comporta, in primo luogo, una fase di training dei weak classifier scelti, utilizzando tutti i dati a disposizione (solitamente viene fornito a ciascun algoritmo/modello di machine learning preso in considerazione un set di training differente).

[57] Il Support Vector Machine è un modello di Supervised machine learning spesso adoperato ai fini di classificazione ma anche di regressione nel campo della computer vision o nel NLP (natural Language processing) - https://lorenzogovoni.com/support-vector-machine/

Algorithm Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H

Figura 3.10[58]: Sintesi del processo di Stacking Ensemble Learning

Successivamente, i modelli allenati effettueranno delle *prediction* su un secondo dataset fornito e tali predizioni saranno adoperate come input per l'addestramento dello *strong classifier*. Questo algoritmo combinatore, sfruttando il lavoro effettuato dai modelli precedenti, riesce a fornire una *final prediction* più accurata di ogni singolo modello preso in considerazione.
Una sintesi del processo di Stacking learning è riassunta in modo esplicativo nella figura [3.11].

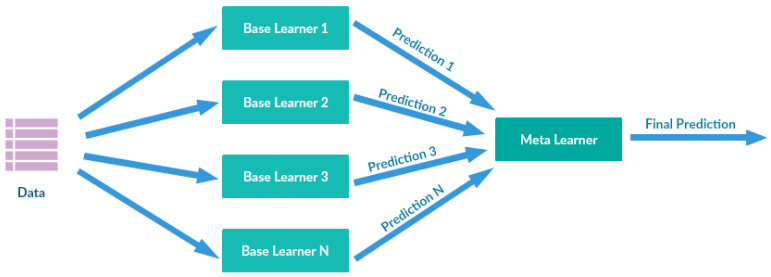


Figura 3.11[59]: Esempio generale di Stacking Ensemble Learning

Dopo questa introduzione teorica relativa allo strumento dell'Ensemble Learning (la metodologia applicata, la tecnica di utilizzo e l'adattamento allo scopo di questa tesi saranno ampiamente descritti nei successivi capitoli) saranno analizzate, nel capitolo successivo, le diverse tipologie di inquadrature cinematografiche, la loro

[58] https://blog.statsbot.co/ensemble-learning-d1dcd548e936

[59] https://mc.ai/icu-survival-prediction-using-ensemble-learning-stacking/

classificazione e l'utilizzo di tale classificazione in ambito creativo. Ciò servirà ad entrare nel vivo dell'analisi sperimentale svolta, che vedrà l'impiego di diversi dataset di immagini e di molteplici modelli innovativi.

Capitolo 4: INQUADRATURE CINEMATOGRAFICHE E LA LORO CLASSIFICAZIONE

4.1 **Introduzione alle inquadrature cinematografiche**

Nel presente lavoro abbiamo scelto come contesto oggetto della sperimentazione le inquadrature cinematografiche, tuttavia l'analisi può trovare applicazione in svariati altri campi, laddove l'obiettivo sia quello di classificare la tipologia delle immagini.

Lo scopo ultimo della tesi, come già accennato nell'introduzione, è dimostrare che è possibile risolvere un problema di chi opera nel campo cinematografico, ovvero il trattamento di una considerevole mole di dati (immagini, video) non strutturati e, quindi, non correttamente organizzati né classificati. Ciò comporta infatti che, soprattutto laddove le immagini e i file video sono molti, l'operatore addetto al montaggio impiega una considerevole quantità di tempo per riorganizzarli poiché dovrà farlo manualmente, con notevole dispendio di energie e un evidente rallentamento dell'attività.

L'esempio dell'operatore del montaggio si attaglia anche ad altre tipologie di attività che comportano l'esigenza di elaborare una grande quantità di dati non strutturati in modo da poter ottenere classificazioni che impattino positivamente sull'efficienza e sulla produttività.

Di qui nasce l'idea di sfruttare le potenzialità dell'apprendimento automatico utilizzando algoritmi in grado di immagazzinare conoscenze da una mole di dati e di operare classificazioni e previsioni sulla base di tali conoscenze. L'obiettivo è quello di applicare metodologie di deep learning non per classificare gli oggetti all'interno delle immagini, bensì le immagini stesse facendo ricorso a tecniche innovative come l'Ensemble Learning, innanzi già illustrato sinteticamente.

A tal fine, in questo elaborato sono state scelte alcune categorie di inquadrature anche se in realtà nell'industria cinematografica se ne contano molte di più.

La sperimentazione, nel nostro caso, ha riguardato otto tipi di inquadrature (*campo lungo, campo medio, mezza figura, mezzo busto, figura intera, piano americano,*

primo piano e primissimo piano), le più elementari e adoperate in ambito cinematografico e quelle che vengono comunemente annoverate dalla letteratura internazionale.

È necessario premettere, per completezza di illustrazione, che l'elenco delle possibili inquadrature è detto scala dei piani ed esse vengono definite in base alla distanza tra la macchina da presa e l'oggetto che viene ripreso[60].

Si parla di "*piano*" quando la macchina da presa si concentra su una figura umana, mentre nel caso in cui l'ambiente o il paesaggio costituiscono la dimensione predominante dell'inquadratura, si ricorre alla denominazione di "*campo*".

Le inquadrature basilari su cui si è incentrata l'analisi vengono di seguito descritte, precisando che il metodo scelto per lo studio (il più comune) è quello della classificazione degli scatti in base alla dimensione del campo (quest'ultima viene determinata dalla parte del soggetto o dell'ambiente mostrata nel campo visivo della telecamera).

4.1.1 *I campi cinematografici*

Il campo lungo (*long shot*), che ha una porzione di ripresa più contenuta rispetto al campo lunghissimo (inquadratura molto simile al long shot che si differenzia solo per un leggero aumento di distanza dell'ambiente dall'obbiettivo e che per tal motivo non è presa in considerazione in questo elaborato), si connota anch'esso per la prevalenza dell'ambiente o del paesaggio sulla figura umana.

Il soggetto non è del tutto assente ma appare solo come un puntino in lontananza oppure, quando si tratta di un insieme di persone, nessuna è distinguibile dall'altra ma si intravede come si intravedono le sue azioni (a differenza del campo lunghissimo in cui la distinzione dei soggetti e delle azioni è pressoché impossibile e per questo è utilizzata solo per mostrare larghe panoramiche). In sostanza la figura umana è collocata nell'ambiente e, pur essendo secondaria, quello che fa è distinguibile, sebbene resti comunque strettamente legata all'ambiente. Dalla figura che segue si evidenzia chiaramente che i personaggi sono inseriti nel paesaggio senza però acquisire un ruolo di preminenza rispetto ad esso.

[60] A. Giaime Alonge, *"Il cinema. Tecnica e linguaggio. Un'introduzione",* Torino, Kaplan, 2011

Figura 4.1[61]**:** Esempio di *"Campo Lungo"*

Nel campo medio (*medium long shot*) l'ambiente prevale ancora ma la figura umana assume un ruolo da protagonista e la sua azione appare molto più evidente. Essa occupa all'incirca la metà dell'inquadratura. È questo il campo usato nei primi film, quelli dei fratelli Lumière, in quanto rispondente a due esigenze: disporre di un campo relativamente ampio (visto che veniva usata una sola inquadratura statica) ma nello stesso tempo non così vasto da non far risaltare la figura umana. La tecnica è praticamente quella utilizzata nel teatro in quanto mira a riprodurre il punto di vista dello spettatore seduto al centro della sala. Come si evince dall'esempio sottostante, la figura umana appare ben riconoscibile poiché è abbastanza vicina, tuttavia lo spazio che le circonda predomina ancora rispetto ad essa.[62]

Figura 4.2[63]**:** Esempio di *"Campo Medio"*

[61] https://www.dreamvideo.it/articoli/221/l-inquadratura-campi-e-piani

[62] https://www.griffithduemila.com/art/tipi-di-inquadrature-cinematografiche-campi-e-piani.html

[63] https://www.dreamvideo.it/articoli/221/l-inquadratura-campi-e-piani

4.1.2 *I piani cinematografici*

Il Piano Medio (P.M.) o Mezza Figura (M.F) [64] prevede l'inquadratura delle figure dalla vita in sù con l'intento di attirare l'attenzione sia sul viso che sui gesti delle braccia del personaggio che risulta ben visibile. L'ambiente viene inquadrato in modo abbastanza chiaro, sebbene solo in parte. Ciò consente di inserire nell'inquadratura anche un secondo personaggio (come da esempio). Nella mezza figura l'ambiente ha perso il suo ruolo predominante mentre l'attore o gli attori riempiono la scena. É l'inquadratura più utilizzata per filmare due personaggi che dialogano, litigano o si abbracciano.[65]

Figura 4.3[66]: Esempio di *"Piano medio"* o *"Mezza Figura"*

Il mezzo busto (M.B.), in inglese *half torso*, è una variante del piano medio. Qui il taglio avviene più o meno all'altezza del petto e viene dato un risalto particolare al volto che diviene protagonista dell'inquadratura.

Ne è un esempio la figura sottostante ove predominano i volti di Brad Pitt e Leonardo Di Caprio in una scena del noto film "*C'era una volta a Hollywood*" di Quentin Tarantino.

[64] *Half figure* – termine anglosassone corrispondente

[65] https://cinemaescuola.files.wordpress.com/2016/09/scala-delle-inquadrature3.pdf

[66] http://www.marellidudovich.gov.it/old/LEZIONI_AUDIOVIDEO/01_inquadratura.pdf

Figura 4.4:[67] Esempio di *"mezzo busto"*

[68]

Nella figura intera (F.I.) il soggetto costituisce l'elemento più importante.
Il personaggio viene inquadrato interamente e copre tutta la verticale dell'immagine. L'ambiente è ancora riconoscibile ma l'attenzione viene concentrata tutta sulla figura umana.[69] Tale inquadratura vene utilizzata soprattutto quando è necessario mostrare l'intero corpo dell'attore, mentre sta compiendo una determinata azione, ad esempio correre o ballare o camminare ecc.

Figura 4.5: Esempio di *"figura intera"*. (Joaquin Phoenix in *Jocker* di Todd Philips)

Nel Piano Americano (*american shot)* il soggetto, generalmente una figura umana, viene inquadrato fino quasi alle ginocchia. Si tratta di una inquadratura molto dinamica, la cui denominazione deriva dal suo grande impiego nei film

[67] https://www.lascimmiapensa.com/2020/02/25/tipi-inquadrature-cinema/2/

[68] In inglese *"Full Shot"*

[69] http://www.cineclub.bz.it/index.php/corso-di-video/377-inquadrature-e-movimenti-di-macchina

western classici. Difatti l'attore viene ripreso dalle ginocchia alla testa poiché è necessario porre in risalto, oltre alla figura umana, anche la pistola, la quale si trova sotto la cintura, e il momento in cui viene estratta dal fodero. Non a caso negli Stati Uniti il Piano Americano è detto anche *cowboy shot* [70].

Figura 4.6[71]: Esempio di *"piano americano"*. (Clint Eastwood in uno *"spaghetti western"* di Sergio Leone)

Il Primo Piano (*close up*) si caratterizza per l'inquadratura della testa e delle spalle del personaggio. Esso mira ad evidenziare ed esprimere la psicologia e le emozioni del soggetto, attraverso la mimica facciale, le espressioni e le parole. Poiché tale inquadratura conferisce un rilievo drammatico all'azione: Inoltre, poiché rileva le tensioni e i sentimenti dell'attore, risponde anche allo scopo di favorire l'identificazione dello spettatore con il personaggio. Il primo piano ha rappresentato una tappa fondamentale nello sviluppo cinematografico del primo decennio del Novecento in quanto ha segnato il superamento delle inquadrature tipiche delle rappresentazioni teatrali (campo medio) e l'avvio di uno stile autonomo sia nella configurazione spaziale, sia nella recitazione.[72]

[70] http://www.cinefile.biz/le-inquadrature-cinematografiche

[71] http://cineclick.utetuniversita.it/linguaggio/inquadratura/piano-americano-/

[72] A. Giaime Alonge, *"Il cinema. Tecnica e linguaggio. Un'introduzione"*, Torino, Kaplan, 2011, cit. nota 56

Figura 4.7[73] **:** Esempio di *"primo piano"*. (Robert De Niro in *The Irishman* di Martin Scorsese)

Nel Primissimo Piano (*extreme close up*) viene inquadrato soltanto il volto del personaggio e a volte viene tagliata anche la parte alta della testa, altre volte pure il mento. Di solito il viso viene ripreso al di sotto dell'attaccatura dei capelli e sino a metà del collo. Questa inquadratura crea una vicinanza particolare tra il personaggio e lo spettatore che ne percepisce quasi i pensieri e riesce a coglierne perfino l'increspatura delle labbra e il battito delle ciglia. L'obiettivo è concentrare l'attenzione di chi guarda su ogni segnale trasmesso dalla espressione del viso dell'attore. Per questo il primissimo piano rappresenta il taglio più psicologico e personale del personaggio[74].

Figura 4.8: Esempio di "*Primissimo piano*". (Jack Nicolson in "*Shining*" di Stanley Kubrick)

Le inquadrature prese in considerazione e descritte poc'anzi, come accennato, non rappresentano la totalità delle tipologie di scatto cinematografico presenti in

[73] https://www.telegraph.co.uk/films/0/irishman-review-netflix-robert-de-niro-sensational-history/

[74] https://www.griffithduemila.com/art/tipi-di-inquadrature-cinematografiche-campi-e-piani.html

letteratura. Per questa prima analisi si è preferito ridurre le classi di inquadrature appositamente, in primis perché lo scopo fondamentale dell'elaborato è quello di verificare la possibilità di classificare tipologie di immagini attraverso metodologie innovative di deep learning (e non gli oggetti al loro interno come viene fatto tradizionalmente) e quindi l'utilizzo di più tipologie di scatto avrebbe esclusivamente aumentato l'entropia all'interno del procedimento di classificazione e la difficoltà di reperimento ed etichettatura dei campioni di dati durante la costituzione dei dataset di analisi (la numerosità dei campioni di ogni classe deve essere omogenea per permettere alla macchina di classificare correttamente e di non cadere in errore. Queste operazioni saranno approfondite nel successivo capitolo). In secondo luogo, alcune classi di inquadrature cinematografiche sono state escluse dal lavoro di analisi per la stretta somiglianza con altre (e.g. campo lunghissimo e campo lungo), prese invece in considerazione, al fine di non creare all'algoritmo problemi di distinzione delle immagini durante il processo di learning con conseguente riduzione dell'efficacia del training e quindi dell'accuratezza della classificazione. Ciò non esclude in prospettiva la possibilità di estendere la sperimentazione anche alle altre tipologie di inquadrature completando così l'intera casistica.

4.2 **Applicazioni della classificazione delle inquadrature cinematografiche attraverso il deep learning**

A seguito dell'approfondimento teorico in merito agli strumenti tecnologici adoperati all'interno dell'analisi e dell'excursus relativo al contesto cinematografico e alle categorie di inquadrature affrontate in questo elaborato, è possibile ora fornire alcuni scenari di applicazione della classificazione, attraverso l'utilizzo del deep learning, degli scatti appena descritti come tassello di un più ampio sfruttamento dell'apprendimento automatizzato (e.g. reti neurali e reti neurali convoluzionali) nel contesto creativo (come già accennato, in questo elaborato viene approfondito l'utilizzo delle tecniche successivamente descritte, nell'ambiente cinematografico però, in caso di necessità, attraverso l'aggiunta di determinate varianti, è possibile adattare gli algoritmi implementati anche per altri

contesti in cui la classificazione della tipologia di immagini, grazie alla pattern detection, può essere utile all'efficientamento del lavoro in modo ottimale).

4.2.1 *Supporto nel video editing*

Come già introdotto nella premessa del capitolo, la grande mole di dati non strutturati (immagini, audio, video...) generati durante la produzione di un film è spesso non organizzata in modo efficiente così da permette all'operatore di montaggio un lavoro ottimale di video editing (i file forniti all'editor sono spesso collezionati in un'unica cartella). Questa disorganizzazione dei file deriva dalle modalità in cui vengono svolte le riprese video che solitamente non hanno un ordine che segue la cronologia delle scene e dell'editing. Cò accade perché sul set, durante la produzione di un film, vengono coinvolti molti individui con ruoli differenti tra cui attori con agende fitte di impegni. Pertanto, in fase di inizio riprese viene data la precedenza alle scene in cui è prevista la partecipazione di più ruoli (ad esempio scene di gruppo in cui sono coinvolti diversi attori, macchinisti e tecnici) così da poter liberare alcuni lavoratori nel momento in cui il loro ruolo non è più necessario al proseguimento delle riprese.
L'operatore di montaggio, in assenza di una corretta classificazione dei file video, non avrà altra scelta che quella di utilizzarli in modo disorganizzato e di cercarli ad uno ad uno per risistemarli in maniera cronologicamente corretta. Ciò implica un certo lasso di tempo laddove i file non sono molto numerosi (e.g. riprese di scene di lunga durata o con tecnica di piano-sequenza[75]) ma nel caso in cui si è in presenza di un consistente numero di video, il montaggio finale delle sequenze secondo la struttura del film diventa un'operazione pressoché impossibile. In questo caso l'operatore avrà un'unica chance: quella di etichettare manualmente i file sobbarcandosi un lavoro organizzativo molto oneroso e che richiede tempi lunghi. L'utilizzo di algoritmi di apprendimento automatico (e.g. reti neurali) consentirebbe invece, mediante l'acquisizione di conoscenze da una grande quantità di dati, di classificare automaticamente le diverse riprese video (grazie alla precedente

[75] *"Il piano sequenza è una lunga inquadratura, quindi senza stacchi, che riprende una o più scene che normalmente sarebbero raccontate con più inquadrature. Si distingue dal long take che invece è una inquadratura della durata inusuale, ma non l'unica all'interno di una determinata scena"*. https://www.cinescuola.it/pianosequenza/

classificazione delle inquadrature cinematografiche, essendo il video un insieme di frame ripresi) semplificando in misura notevole l'attività dell'editor. Immaginiamo il caso di un operatore che alla fine di una giornata in cui sono state girate le diverse riprese, debba svolgere il montaggio della pellicola. Egli dispone solo dello storyboard ma deve rimettere a posto la cronologia delle riprese stesse secondo le indicazioni del regista: se non può servirsi di un tool tecnologico che gli permetta di incasellare i file video nelle rispettive classi, impiegherà ore o giorni per giungere al risultato finale.

4.2.2 *Ottimizzazione del trailer making*

Il deep learning può essere utilizzato anche per analizzare modelli di trailer di film nei componenti audio e visivi con l'obiettivo di individuare le qualità che rendono un trailer attrattivo per lo spettatore. In sostanza, combinando l'analisi dei trailer con altri dati (ad esempio le informazioni storiche sulle preferenze dei clienti che il produttore ha a disposizione) è possibile migliorare la previsione del comportamento dell'utente. Gli algoritmi[76]di deep learning verranno usati per classificare le immagini cinematografiche, le scene girate e le note audio e il risultato finale sarà elaborato, tenendo conto anche degli altri dati disponibili (scelte effettuate dallo spettatore), in un arco di tempo molto più breve di quello normalmente necessario per produrre un trailer con il metodo tradizionale. Per entrare nel dettaglio, l'algoritmo viene addestrato a classificare una serie di inquadrature cinematografiche relative a trailer di film contenenti alberi, volti, auto, animali, colore, illuminazione ecc. (la classificazione delle immagini cinematografiche è un tassello importante del cruscotto di strumenti e algoritmi di deep learning che permettono questa operazione) , colonne sonore, ne registra la frequenza e il tempo di permanenza sullo schermo, poi li associa al genere di film (ad esempio una lunga inquadratura di un volto sarà certamente appartenente a un genere drammatico) e impara a gestire tutte le informazioni acquisite per giungere a determinare le scene più suggestive di un trailer, quelle che hanno maggiormente catturato l'attenzione dello spettatore e generato emozioni positive. L'algoritmo

[76]https://www.tomshw.it/altro/film-e-trailer-tutti-uguali-con-lintelligenza-artificiale-potrebbe-andare-peggio/

sarà addestrato su trailer di ottima fattura relativi a film famosi e imparerà a riconoscere le immagini, i video e le colonne sonore più utilizzate dai migliori montatori trailer in modo che riescano ad estrarre le sequenze video più efficaci e di impatto di nuovi film, basandosi su vecchi trailer di successo.
Un esperimento simile è stato condotto da IBM per la creazione del trailer del thriller "Morgan". Il colosso informatico, con il supercomputer Watson, ha esaminato cento trailer di film horror e ne ha ricavato un elenco di dieci scene, per un totale di sei minuti, ritenendole le migliori per un trailer. Il video editor ha impiegato solo 24 ore per dare vita al trailer del film, a fronte dei dieci/trenta giorni di regola necessari.[77]

4.2.3 *Riconoscimento del contenuto, del genere e della paternità di un film per la smart reccommendation*

Altra possibile applicazione della classificazione delle inquadrature cinematografiche può essere, come introdotto già nel precedente scenario, quella utile a riconoscere il genere di film, il contenuto o anche l'autore. Le inquadrature contenute in un film hanno tutte una loro funzione narrativa (e.g. la distanza della telecamera ha un impatto molto influente e profondo sul coinvolgimento e le emozioni trasmesse al pubblico e soprattutto sull'identificazione con i protagonisti del film[78]) e spesso alcuni generi di film adoperano più frequentemente, durante le riprese, tipologie specifiche di inquadratura (e.g. in un film western sono utilizzati spesso scatti come il campo medio per permettere allo spettatore di vedere le mosse di entrambi i pistoleri durante un duello e come il piano americano per catturare il momento dell'estrazione della pistola). Questa classificazione automatizzata delle inquadrature, coadiuvata da altri algoritmi di deep learning in grado di lavorare con i colori o le note audio (e.g. colonne sonore, canzoni...) potrebbe permettere di riconoscere il genere di un film con una buona accuratezza. Oltre al genere della pellicola, le inquadrature utilizzate caratterizzano anche lo stile di un determinato

[77] https://creativefuture.co/artificial-intelligence-automation-film-video-machine-learning-editing-robots-cinematography/

[78] Benini, Sergio & Canini, Luca & Leonardi, Riccardo. (2010). Estimating cinematographic scene depth in movie shots. 2010 IEEE International Conference on Multimedia and Expo, ICME 2010. 855 - 860. 10.1109/ICME.2010.5582611

regista che può essere individuato dalla frequenza e dalla sequenza con cui utilizza certi scatti. Il grande regista Stanley Kubrick, ad esempio, diceva *"cinema è close up"* [79] data la sua passione per il primo piano che gli permetteva in film, come ad esempio *Shining*, di creare un legame emotivo tra protagonista e spettatore, focalizzarne l'attenzione su un particolare e renderlo più partecipe alla scena. È evidente dunque che una classificazione automatica delle inquadrature tramite apprendimento profondo può rendere semplice l'attribuzione della paternità di un determinato film ad un regista in modo efficiente e accurato[80].

Le classificazioni e i riconoscimenti automatici appena introdotti si sono rivelati efficienti per diverse applicazioni come l'indicizzazione dei film e quindi la possibilità di implementare la *smart reccommendation*[81].

Con il grande aumento di film e serie tv attualmente resi disponibili sulle piattaforme on demand streaming (e.g. Netflix, Amazon Prime Video ecc.), la scelta di cosa guardare diventa sempre più importante e quindi la raccomandazione personalizzata, creata ad hoc per lo spettatore, costituisce un elemento fondamentale per l'esperienza di intrattenimento del pubblico. Un grande problema per queste piattaforme è dunque proprio la modalità con cui consigliare ai clienti ciò che di meglio offrono i loro cataloghi in base alle preferenze soggettive e il successivo miglioramento delle raccomandazioni nel tempo. Le tecniche attuali di smart reccommendation utilizzano la cronologia di visualizzazione del cliente e il rating fornito da altri clienti o da esperti del settore, questo però porta a risultati poco personalizzati perché la raccomandazione più probabile sarà quella del film o della serie tv più vista e famosa e non il contenuto digitale più adatto al cliente.

L'estrazione di informazioni e caratteristiche delle scene come i tipi di inquadrature cinematografiche, colori e colonne sonore presenti attraverso il deep learning e i successivi procedimenti di riconoscimento del genere, del contenuto (e.g. lo svolgimento della pellicola prettamente in un luogo chiuso o predominanza di lunghi dialoghi come nei film di Woody Allen) e dell'autore permettono di superare gli attuali problemi di raccomandazione personalizzata creando un'indicizzazione

[79] http://polisemantica.blogspot.com/2016/04/il-primo-piano-linquadratura-prediletta.html

[80] Svanera, Michele et al. "Who Is the Film's Director? Authorship Recognition Based on Shot Features." IEEE MultiMedia 26.4 (2019): 43–54. Crossref. Web

[81] D. Ciresan, U. Meier, and J. Schmidhuber. *Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition* (CVPR), 2012 IEEE Conference on, pages 3642–3649. IEEE, 2012.

perfetta e a tutto tondo delle pellicole. L'indicizzazione poi permetterà al cliente di effettuare ricerche video efficienti[82] (e.g. Il cliente sarà in grado di trovare film anche solo fornendo al dispositivo parole come "Nolan – thriller- in una caverna" per ricercare tutta la saga DC Comics di Batman) e i dati generati da queste ricerche e dalle visualizzazioni dello spettatore (informazioni dettagliate sui film, l'ambientazione, l'autore, le tipo di riprese video) renderanno semplice la creazione di un sistema di *suggestion* ad hoc[83]. Le potenzialità derivanti dalla raccomandazione personalizzata basata sul riconoscimento degli autori, dei generi e del contenuto dei film sono enormi e avranno un grande impatto sull'esperienza di ricerca della pellicola perfetta per il cliente e quindi sulla *entertainment experience* degli utenti.

[82] D. C Cireşan, U. Meier, J. Masci, L. M Gambardella, and J. Schmidhuber. *High-performance neural networks for visual object classification*. arXiv preprint arXiv:1102.0183, 2011.

[83] S. Arora, A. Bhaskara, R. Ge, and T. Ma. *Provable bounds for learning some deep representations*. arXiv preprint arXiv:1310.6343, 2013.

Capitolo 5: TRATTAMENTO DATI E SPERIMENTAZIONE

L'obiettivo che si prefigge la fase di sperimentazione di questo elaborato è quello di classificare in modo accurato le inquadrature più usate nel mondo del cinema (precedentemente descritte), adoperando una metodologia innovativa di Deep Learning che migliori i risultati già raggiunti da altri con sperimentazioni aventi lo stesso scopo. Tale classificazione "*intelligente"*, in sinergia con altri algoritmi di deep learning applicati al mondo del cinema (riconoscimento delle colonne sonore o dei colori...), risulta essere estremamente importante al fine di costruire software con diverse applicazioni a supporto degli "addetti ai lavori" in diversi campi come: il montaggio video, la riorganizzazione di set fotografici, l'ottimizzazione del trailer making e l'implementazione della smart reccommendation dei contenuti multimediali streaming .

Lo strumento utilizzato è l'Ensemble Learning con approccio Stacking, il cui funzionamento teorico è stato illustrato nel capitolo precedente. Questa tecnica è stata adoperata per combinare e migliorare i risultati di classificazione di tre reti neurali convoluzionali di tipologia VGG16[84], tutte con medesima architettura e di base addestrate sul dataset "*ImageNet"* ma con diversi layer ri-trainati attraverso il transfer learning su tre dataset appositamente costruiti (questa modalità è stata scelta perché permette di sfruttare appieno la conoscenza e l'accuratezza pregressa della rete. Infatti, ri-allenando esclusivamente gli ultimi layer al fine di adattare la CNN a svolgere il nostro scopo, non si ha la necessità di grandi set di dati che richiederebbero grande effort per il reperimento). Il primo dei tre set di dati è stato creato da zero, non esistendo collezioni di immagini già ordinate nelle classi di inquadrature desiderate, con l'utilizzo di diversi algoritmi in linguaggio Python implementati per effettuare l'estrazione delle immagini, il Preprocessing e

[84] La rete VGG16 è una famosa CNN introdotta da *K.Simonyan* e *A.Zisserman* dell'università di Oxford all'interno dell'articolo scientifico *"Very Deep Convolutional Networks for Large-Scale Image Recognition"*. Questa rete deve il suo successo al traguardo raggiunto nei test di accuretezza di classificazione del dataset *"ImageNet"* composto da più di 15 milioni di immagini ad alta risoluzione reperite su internet e suddivise in più di 1000 classi. https://neurohive.io/en/popular-networks/vgg16/

l'ordinamento nelle rispettive classi. Esso è composto da immagini a colori (RGB[85]) che prima di essere date in input alla CNN, hanno subito un iter di Preprocessing al fine di renderle ottimali per l'addestramento del modello. Ad esempio, prima della fase di ordinamento, le immagini sono state tutte allineate ad una risoluzione di 320x180 pixel in modo da omogenizzare il dataset ad una risoluzione processabile dalla macchina e permettere una visualizzazione ottimale e un facile riconoscimento durante l'etichettatura. Successivamente, a monte dell'allenamento della rete, la risoluzione è stata portata a 160x90 e in formato 16:9 per due motivi ben precisi. Il primo è di tipo computazionale poiché un computer, nel momento in cui riceve un immagine a colori in input, la legge come un insieme di valori da 0 a 255 (0 corrisponde al nero assoluto e 255 al bianco) su tre scale RGB (rossa, verde e blu) corrispondenti ad un pixel da illuminare e che, combinate, indicano un colore ben preciso. Per il calcolatore ogni pixel è una caratteristica da processare, quindi per la risoluzione 160x90 su tre canali di colore, il numero di feature è 160x90x3, uguale a 43.200 caratteristiche. Il numero è già molto elevato per il carico computazionale e quindi in caso si voglia aumentare la risoluzione si dovrà mettere in conto un evidente rallentamento del processo di calcolo. L'altra motivazione che ha portato a scegliere la risoluzione 160x90 e non una minore (che avrebbe ridotto ancora il carico computazionale) o maggiore, è il rapporto 16:9, molto utilizzato nel mondo del cinema nell'ultima decade e formato principale delle immagini e dei video ad alta risoluzione. L'utilizzo di tale formato ha pertanto reso più semplice il reperimento di immagini in campo cinematografico e, nello stesso tempo, consentirà una più facile applicazione pratica del presente lavoro di tesi nel campo creativo e nella creazione di un *tool* di supporto agli addetti ai lavori.

Nel paragrafo dedicato al dataset RGB verranno approfondite le fasi del Preprocessing, quelle successive come il *white balance*[86] (che consente di eliminare qualsiasi alterazione innaturale dei colori delle immagini dovuta alla differenza di esposizione, nonché ad uniformare alla stessa tonalità i campioni di dati provenienti

[85] RGB corrisponde alla sigla di un modello di colori additivo che combina il rosso, il verde e il blu (da qui l'acronimo: Red, Green, Blue) per creare la maggior parte delle sfumature di colore percepibili dall'occhio umano e rappresentabili dai calcolatori. Per questo motivo è molto utilizzato in ambito elettronico-informatico: https://it.wikipedia.org/wiki/RGB

[86] Il bilanciamento del bianco è la procedura grazie alla quale siamo in grado di interpretare in modo corretto la "luce della scena" e ottenere una scala di colori veritieri. L'operazione si basa sulla temperatura colore di una sorgente di luce e a valle di questa, è possibile eliminare sfumature innaturali dei colori e risalire alla colorazione originale dell'immagine https://www.xn--photocaf-80a.it/bilanciamento-del-bianco/

da fonti diverse e cogliere le reali colorazioni delle immagini) e l'ordinamento delle immagini.

Gli altri due dataset sono stati generati a partire dal primo a valle del Preprocessing e dell'etichettatura. Il secondo dataset che d'ora in poi chiameremo *Image Segmentation* dataset o dataset stilizzato (la semantic image Segmentation verrà descritta in un apposito paragrafo), è frutto dell'algoritmo di segmentazione semantica *DeepLab V3 Plus*[87] con cui vengono processate le immagini a colori presenti nel dataset RGB già ordinato, con lo scopo di generare un set di dati in cui l'informazione visuale è segmentata e le immagini sono stilizzate e contengono gli oggetti ben delineati (gli elementi fondamentali dell'immagine sono messi in risalto per un facile riconoscimento da parte della rete). Il terzo e ultimo set di dati invece è sempre basato sul dataset RGB ed è un'estrazione, tramite algoritmo Python, delle Ipercolonne delle immagini (d'ora in poi, *Hypercolumns Dataset*) che ci ha fornito un set di dati in cui sono esaltate le caratteristiche basilari di ogni immagine.

A valle della presentazione e dell'approfondimento dei set di dati costituiti per questa analisi è inserita una fase di *data exploration* la quale evidenzia la complessità del problema affrontato generata dalla grande dimensionalità dei dati. Tale esplorazione è attuata tramite l'applicazione di algoritmi di *dimensionality reduction* e Unsupervised machine learning come il clustering (così da rendere più chiaro al lettore il traguardo raggiunto con il Deep Learning nell'affrontare il problema).

5.1 **Creazione dei set di dati e Preprocessing**

Nella seguente sezione approfondiremo, a livello pratico, il trattamento dei dati che comprende le fasi di estrazione dei campioni, Preprocessing, etichettatura, creazione dei tre dataset introdotti attraverso algoritmi e modelli implementati in Python e infine la realizzazione di tre training set per le nostre *fine tuned VGG16*.

[87] "Il DeepLab è un modello di Deep Learning allo stato dell'arte per la segmentazione semantica dell'immagine, dove l'obiettivo è quello di assegnare etichette semantiche (ad esempio, persona, cane, gatto e così via) ad ogni pixel dell'immagine in ingresso" https://supervise.ly/deep-lab-v-3-plus/overview

5.1.1 *Dataset 1: RGB*

Il set di immagini in questione è la base dei successivi dataset estratti attraverso modelli di deep learning, a valle però di tutte le operazioni che ora andremo a descrivere in dettaglio. Questa collezione di immagini è stata creata da zero non esistendo su internet un database di campioni suddivisi nelle diverse inquadrature cinematografiche (la motivazione, in primis, è che le classi di inquadrature, come già detto, non sono univoche nelle varie nazioni, in secondo luogo gli studi in merito all'utilizzo delle tecniche di Deep Learning nel campo cinematografico sono ancora ad uno stato embrionale e quindi ancora nessuna casa produttrice ha commissionato un lavoro di collezionamento dei campioni su larga scala).
Le immagini a colori che compongono questo dataset hanno fonti diverse.
La maggior parte sono state collezionate tramite un algoritmo di estrazione dei frame video implementato su video YouTube di trailer famosi (questo algoritmo è descritto nel successivo paragrafo), altre sono state accuratamente selezionate da diversi database presenti sul portale Kaggle[88] (face recognition, COVID-19 mask presence recognition, Landscape...) e le restanti sono scatti di fotografi professionisti o amatoriali. Proprio per questo motivo, prima di fornire in input al modello le immagini molto differenti tra loro a livello di risoluzione, di bianchi, di esposizione, rapporto d'aspetto, formato ecc.., è stato necessario prevedere un iter di Preprocessing delle stesse, per permettere alla rete neurale convoluzionale VGG16 di poter apprendere nel modo più efficiente e veloce ed essere in grado di generalizzare la conoscenza acquisita senza commettere errori. Il primo step effettuato è stata la riduzione di tutte le immagini ad un'unica dimensione, risoluzione (stesso numero di pixel in orizzontale e verticale) e *aspect ratio* perché in caso la CNN riceva immagini con dimensioni differenti, il processo di addestramento della rete può bloccarsi già al primo layer per problemi di elaborazione (la dimensione dei dati va dichiarata in input alla rete e deve essere fisso). Il passo successivo è stato l'utilizzo di un algoritmo in Python di *White Balance* in modo da ristabilire la colorazione originale e, nella terza fase, sono state etichettate e ordinate le varie immagini delle diverse classi mediante un semplice script in Python. Nell'ultima fase di questo processo di "*Data Quality"*, durante la

[88] https://www.kaggle.com/

creazione del training set, le immagini sono state tutte ridimensionate ulteriormente per ridurre il carico computazionale. A seguire vengono approfonditi e meglio descritti tali step e i relativi algoritmi adoperati.

5.1.1.1 Estrazione dei frame video

Come già anticipato, il primo passo per la costruzione del dataset RGB è stato quello di reperire e collezionare le immagini da diverse fonti. Oltre alla selezione di campioni da altri dataset presenti su internet, i dati sono stati frutto di un'estrazione di frame video derivanti da trailer presenti sulla piattaforma web di contenuti multimediali YouTube. Il funzionamento dello script è molto semplice: viene fornito un URL di un video presente su YouTube e l'algoritmo estrae un fermo immagine ogni venti secondi in modo da non catturare immagini troppo simili (i secondi per ogni cattura sono modificabili in base al video da segmentare).

```
import cv2
import os[89]
import shutil
import pafy

destination='C:/Users/Francesco/Desktop/Video_Frame_Capture'
current_frame=0
url='www.youtube.com/Warner Bros/Harry Potter_I doni della morte_trailer'
video_Pafy=pafy.new(url)
play_video=video_Pafy.getbestvideo(preftype='webm')
video=cv2.VideoCapture(play.url)
```

In questo primo estratto del codice di frame capture vengono importate le librerie principali che hanno lo scopo di interagire con la piattaforma streaming, catturare il video, scrivere il file e infine spostarlo nella cartella di destinazione dichiarata nella linea di codice successiva. A seguire viene determinato il frame da cui partire che di default è il primo (frame=0), questo frame verrà incrementato durante il

[89] https://opencv-python-tutroals.readthedocs.io/py_tutorials/py_video_display.html

processo di capture per estrarre un immagine ogni n secondi. Successivamente viene dichiarato l'URL che caratterizza il video che vogliamo sezionare e attraverso le funzioni della libreria Open CV e Pafy[90] e del backend YouTube-dl di cui si avvale (installati custom con la funzione PIP) si crea un'istanza video dall'url di YouTube e lo si seleziona alla migliore risoluzione (che successivamente verrà abbassata per i motivi computazionali già spiegati) e ad un determinato formato (e.g. webm) ma senza audio.

```
try:
    if not os.path.exists('destinazione'):
        os.makedirs('destinazione')
except OSError:
    print ('Error: Creating directory of data')

while(True):
    ret, frame=video.read()
    if ret:
        if current_frame%20==0:
            name='./destinazione/Trailer_2020Ita_'+str(current_frame)+'.jpg'
            print("Creating..." + name)
            cv2.imwrite(name, frame)
            shutil.move(name, dest)
            current_frame+=1
        else:
            current_frame+=1
    else:
        break

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

In queste ultime linee di codice viene prima verificata l'esistenza della cartella di destinazione e, solo in caso non risulti in locale, viene comunque generata prima di

[90] https://pythonhosted.org/Pafy/

qualsiasi altra operazione in modo da poter continuare il processo di cattura (è stata inserita, tramite i comandi Python Try and Except, anche la gestione dell'eccezione in caso la creazione non vada a buon fine). In seguito, nel ciclo while, i frame dei video sono letti e viene dichiarato il nome del file immagine, arricchito della desinenza che indica il formato, in modo progressivo grazie al numero di frame incrementato ad ogni cattura. In conclusione, viene scritto il file e spostato nella cartella di destinazione corretta. Il ciclo di cattura si chiude al termine del video selezionato.

Figura 5.1: Esempi di frame estratto attraverso l'algoritmo descritto. fonte: 'www.youtube.com/Warner Bros/Harry Potter_I doni della morte_trailer'

5.1.1.2 Ridimensionamento delle immagini e conversione di aspect ratio

A valle della collezione del capitale multimediale e quindi dopo aver creato un dataset non ordinato di immagini RGB, lo step successivo dell'iter di Preprocessing è consistito nell'effettuare un ridimensionamento delle immagini ad una risoluzione minore (320x180) adatta ad una facile visualizzazione per l'occhio umano e interpretazione durante l'etichettatura tramite algoritmo in Python ma comunque ridotta a livello di carico computazionale, in confronto all'alta risoluzione della versione originale. Oltre a questa operazione principale, è stata effettuata anche la conversione di tutto il set di dati ad un aspect ratio unico di 16:9. Come già ampiamente descritto, questo formato è stato scelto perché di grande utilizzo nel mondo del cinema e della televisione e inoltre perché è stato dimostrato che questo tipo di rapporto d'aspetto è quello a cui è più semplice ricondurre qualsiasi altra immagine con formato differente, senza alterare la rappresentazione. Proprio per non alterare l'immagine, la conversione a 16:9 è stata operata ritagliando le immagini (per conservare le proporzioni, in alcuni casi, sono stati tagliati i bordi dell'immagine che però non avrebbero fornito informazione ulteriore) e non allungandole (*stretching*), questo perché un'immagine molto alterata, anche se utile

ad aumentare il campione di addestramento della rete e quindi l'eventuale accuratezza, fornirebbe un sample irrealistico, non applicabile al campo cinematografico e, quindi, falserebbe i risultati ottenuti. Per questa ragione sono state scartate immagini con rapporti d'aspetto troppo differenti dal 16:9 che avrebbero richiesto uno *stretching* per ricondurle ad esso (e.g. immagini verticali). Gli step appena descritti sono stati effettuati tramite il software di manipolazione digitale Photoshop, creando un blocco di operazioni sequenziali applicabile ad un vasto pool di immagini in modo semplice e rapido. Applicati tali step a tutte le immagini si è passati all'ultima fase: l'ordinamento finale del dataset nelle varie classi (eseguito grazie ad uno script in Python) e l'operazione di bilanciamento del bianco classe per classe. A valle di questa operazione si è passati alla creazione del training set per la prima rete VGG16 e all'estrazione degli altri set di dati, già ordinati e in qualità.

5.1.1.3 Etichettatura delle immagini

Dopo aver creato una prima collezione di dati non ancora riordinata, comprendente immagini di ogni tipologia di inquadratura, si è reso necessario etichettare tutti i campioni nel modo più corretto possibile nelle rispettive classi così che questi label possano essere utilizzati per l'addestramento della rete e il calcolo dell'accuratezza. Questa operazione di ordinamento deve necessariamente essere effettuata conservando le proporzioni in termini di numerosità per classe. Il numero di immagini per categoria di inquadratura deve essere il più possibile equilibrato in maniera tale che la rete neurale, in fase di apprendimento, non noti una prevalenza di campioni per una certa categoria e la utilizzi come scappatoia in caso si trovi in difficoltà nella classificazione (avendo valore in percentuale sul totale più elevato, una classe viene scelta come la più probabile in ogni situazione in cui la rete non sa porre un'etichetta al campione). Tale procedura è stata implementata tramite un algoritmo che rende l'operazione decisamente più veloce di uno smistamento manuale. Lo script, in sostanza, legge l'immagine, la mostra all'utente che in un semplice click può etichettarla correttamente secondo la sua percezione. Naturalmente il processo di etichettatura è simulato, in realtà l'algoritmo, all'input dell'utente, sposta l'immagine in una cartella appositamente creata con il nome

dell'inquadratura. Questo perché durante la fase di creazione del training set, l'etichetta verrà estratta proprio dalla cartella di appartenenza dell'immagine.

```
import os
import cv2
import matplotlib.pyplot as plt

from pathlib import Path
import shutil

Campo Lungo='C:/Users/Francesco/Desktop/DatasetRGB/CL'
Campo Medio='C:/Users/Francesco/Desktop/ DatasetRGB/CM'
Mezzo Busto='C:/Users/Francesco/Desktop/ DatasetRGB/MB'
Mezza Figura='C:/Users/Francesco/Desktop/DatasetRGB /MF'
Figura Intera='C:/Users/Francesco/Desktop/ DatasetRGB/FI'
Piano Americano='C:/Users/Francesco/Desktop/ DatasetRGB/PA'
Primo Piano='C:/Users/Francesco/Desktop/DatasetRGB/PP'
Primissimo Piano='C:/Users/Francesco/Desktop/ DatasetRGB/PPP'
cestino='C:/Users/Francesco/Desktop/ cestino'
```

In questa prima estrazione dello script avviene l'import delle librerie adoperate per leggere le immagini tramite il path fornito, manipolarle, mostrarle all'utente e poi permettere lo spostamento intercartella dopo la scelta della classe. A seguire invece osserviamo i percorsi caratterizzanti le cartelle di destinazione in cui verranno smistate le immagini. Come si può notare è fornito un path per ogni inquadratura e ogni cartella si trova all'interno di una più generale chiamata "DatasetRGB". É stata inserita anche una cartella-cestino come destinazione di quelle immagini non appartenenti a nessuna classe e quindi da scartare.

```
percorso=Path('C:/Users/Francesco/Desktop/Shuffle_DatasetRGB')
for image in os.listdir(percorso):
   try:
      image_path=os.path.join(percorso,image)
      image_array=cv2.imread(imege_path, cv2.IMREAD_UNCHANGED)
```

```
imege_array=cv2.cvtColor(image_array, cv2.COLOR_BGR2RGB)
plt.imshow(image_array, cmap="gray")
plt.show()
```

Entrando nel vivo del codice, la prima parte legge il path che caratterizza le immagini da etichettare, queste attraversano una alla volta un ciclo in cui vengono manipolate in modo tale da renderle di facile visualizzazione all'utente e permettere la scelta (e.g. la libreria Open CV, nel momento in cui legge le immagini, non utilizza le normali scale di colore Red, Green e Blue ma per convenzione inverte i canali in Blue, Green e Red, perciò, per mostrare l'immagine all'utente in modo corretto, i dati vanno riconvertiti allo spazio colore RGB). A seguire quindi l'immagine viene mostrata all'utente tramite le funzionalità di *show* della libreria MathPlotLib, il ciclo si blocca al terminare delle immagini all'interno della cartella sorgente (il codice è inserito tra i già introdotti Try and Except che gestiranno l'eccezione di errore in lettura del file generando un allert di errore e passando all'immagine successiva senza bloccare il processo).

```
Input=int(input("inserire il numero 1 per campo lungo, 2 per campo medio, 3 per figura intera, 4 per piano americano, 5 per mezza busto, 6 per mezza figura, 7 per primo piano, 8 per primissimo piano, 0 per terminare l'algoritmo, 9 per eliminare l'immagine: "))

print(Input)

    if Input==1:
        print("Campo Lungo")
        shutil.move(image_path, Campo Lungo)

    elif Input==2:
        print("Campo Medio")
        shutil.move(image_path, Campo Medio)

    elif Input==3:
        print("Figura Intera")
```

```
            shutil.move(image_path, Figura Intera)

        elif Input==4:
            print("Piano Americano")
            shutil.move(image_path, Piano Americano)

        elif Input==5:
            print("Mezza Busto")
            shutil.move(image_path, Mezzo Busto)

        elif Input==6:
            print("Mezza Figura")
            shutil.move(img_path, Mezza Figura)

        elif Input==7:
            print("Primo Piano")
            shutil.move(image_path, Primo Piano)

        elif Input==8:
            print("Primissimo Piano")
            shutil.move(image_path, Primissimo Piano)

        elif Input==9:
            print("immagine eliminata")
            shutil.move(image_path,cestino)

        elif Input==0:
            print("Programma terminato")
            break
    except Exception as e:
        print("errore")
    pass
```

L'estratto finale del codice mostra la fase in cui l'utente può effettuare l'operazione di ordinamento apponendo un label alle immagini mostrate che verranno spostate nelle relative cartelle corrette. In caso di necessità l'utente può anche eliminare l'immagine o terminare il processo. Questa fase è fondamentale per tutta la metodologia di classificazione delle inquadrature perché un buon capitale di immagini ben etichettate da dare in input alla rete in fase di addestramento permette di raggiungere un'accuratezza più elevata (la rete con un training set ordinato in modo corretto riesce a riconoscere meglio i pattern che contraddistinguono ogni classe e a generalizzare i risultati senza commettere errori grossolani dovuti a campioni etichettati nella classe sbagliata).

5.1.1.4 Bilanciamento del bianco

La white balance procedure, operazione che permette di interpretare nel modo più corretto l'illuminazione della scena e di ottenere una colorazione veritiera più attinente a quella originale, è stata applicata per eliminare qualsiasi alterazione della tonalità dei colori delle immagini all'interno del dataset RGB (alcune immagini, provenendo da fonti diverse, potrebbero essere state già sottoposte a post production con strumenti di photo o video editing). La necessità deriva dalla modalità già descritta, con cui le reti leggono le immagini (tre scale di colori dove ogni pixel ha un valore da 0 a 255, in cui 0 è il nero e 255 corrisponde proprio al bianco, e il valore e la proporzione di queste scale determina il colore): Infatti, con valori alterati dei bianchi le reti interpreterebbero le immagini in modo non corretto, portando a risultati con un'accuratezza ridotta (se la rete viene allenata su immagini con tonalità di colori alterate, nel momento in cui vengono forniti campioni mai processati, questa fornirà interpretazioni errate).
L'operazione di bilanciamento dei bianchi è stata effettuata tramite un semplice script in linguaggio Python [91] che verrà descritto qui in dettaglio.

```
import cv2
import os
```

[91] https://stackoverflow.com/automatic-white-balancing-with-grayworld-assumption

```
import matplotlib.pyplot as plt
import numpy as np

def white_balance_loops(image):
    result = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    average_a = np.average(result[:, :, 1])
    average_b = np.average(result[:, :, 2])
    for x in range(result.shape[0]):
        for y in range(result.shape[1]):
            l, a, b = result[x, y, :]
            l *= 100 / 255.0
            result[x, y, 1] = a - ((average_a - 128) * (l / 100.0) * 1.1)
            result[x, y, 2] = b - ((average_b - 128) * (l / 100.0) * 1.1)
    result = cv2.cvtColor(result, cv2.COLOR_LAB2BGR)
    return result
```

Nell'estratto di codice presentato viene implementata la funzione di bilanciamento del bianco in cui viene fornita un'immagine alla volta come parametro in input. Questa, all'interno della funzione, subisce diverse operazioni di *image processing, color processing* e *array transformation* tramite l'utilizzo di funzione delle librerie Open CV[92] e Numpy. In prima battuta, l'immagine subisce una conversione dal modello di colore BGR allo spazio colore LAB[93] con cui è più efficiente lavorare per effettuare le operazioni di white balance. A seguire l'immagine viene manipolata come Numpy array per cui viene estratto il valore medio per il secondo e terzo asse di colore (gli assi che determinano la dimensione del colore). Infine, attraverso diverse operazioni di calcolo (normalizzazione dei pixel...) sulla dimensione 0 e 1 dell'array vengono bilanciati i bianchi e l'immagine viene riconvertita nel formato BGR leggibile dalle funzioni Open CV.

source_path='/'C:/Users/Francesco/Desktop /DatasetRGB/Piano Americano'

[92] https://machinelearningknowledge.ai/opencv-tutorial-image-colorspace-conversion-using-cv2-cvtcolor

[93] Il modello LAB è uno spazio colore in cui L è la variabile che corrisponde alla luminosità calcolata attraverso la radice cubica della luminanza relativa e A e B alle dimensioni del colore in coordinate https://it.wikipedia.org/wiki/Spazio_colore_Lab

```
destination_path='/'C:/Users/Francesco/Desktop/WB_DatasetRGB/Piano
Americano'

count=0
for image in os.listdir(source_path):
   try:
      count+=1
      image=cv2.imread(os.path.join(source_path,image), cv2.IMREAD_COLOR)
      image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
      image=white_balance_loops(image)

      plt.imshow(image)
      plt.savefig(os.path.join(destination_path,(f'PA{count}.jpg'))

   except Exception as e:
      print("error")
      pass
```

Naturalmente, dopo aver dichiarato la funzione di white balance, questa va utilizzata fornendole in input le immagini da trasformare. Per questo motivo attraverso un ciclo for vengono aperte le immagini (il ciclo costruito è eseguito per ogni cartella corrispondente ad un'inquadratura) e passate alla funzione di bilanciamento bianco che restituirà l'immagine trasformata. Questa poi viene salvata in una cartella apposita e il nome sarà composto da un contatore che aumenta ad ogni loop del ciclo, dal nome della classe e dal formato desiderato.

5.1.2 *Dataset 2: Image Segmentation*

La Semantic Image Segmentation[94] ha lo scopo di etichettare tutti i pixel di un'immagine con una relativa classe di appartenenza e quindi categorizzare e suddividere i pixel delineando curve, angoli ed elementi fondamentali di un oggetto

[94] https://www.jeremyjordan.me/semantic-segmentation/

presente all'interno attraverso predizioni effettuate su ogni pixel contenuto nel campione (per questo Semantic Image Segmentation è spesso chiamata con l'appellativo di Dense Prediction). Spesso, per convenzione ed interpretabilità, le etichette vengono fatte corrispondere a dei colori dello spazio RGB (e.g. il giallo per le persone, il rosso per le auto...) creando delle mappe di predizioni. Essendo il dataset di partenza una collezione molto complessa di dati (come vedremo nel capitolo di model e data explanation, il problema considerato ha una complessità molto elevata ereditata proprio dai dati di partenza) questa procedura è stata scelta per la creazione del secondo dataset perché ha l'abilità di rendere le immagini semplificate e la loro rappresentazione più significativa e analizzabile[95].

La semplificazione dei dati potrebbe essere molto utile per mettere in risalto oggetti e figure presenti all'interno delle immagini, determinanti per la scelta di una classe di inquadrature (e.g. ben delineare la figura umana nell'immagini può portare già ad escludere alcune inquadrature come il campo lungo e, in base alla vicinanza all'obbiettivo, classificare tra figura intera, mezza figura, primo piano ecc..).

La semantic image Segmentation però è implementabile con diverse reti neurali che forniscono segmentazioni e quindi output differenti, più o meno utili allo scopo di classificare le inquadrature. In questo elaborato sono state sperimentate tre reti per la segmentazione delle immagini e la scelta della migliore è stata guidata dall'output risultante dal modello in termini di messa in risalto delle figure umane e dall'interpretabilità che una VGG16 riusciva a dare a queste immagini segmentate (l'interpretabilità è stata valutata effettuando prove di allenamento della VGG16 con piccoli campioni di immagini ordinate, generate dalle tre diverse reti e confrontando l'accuratezza di classificazione). In primo luogo, a fronte della moltitudine di librerie e backend con cui implementare la semantic Segmentation (e.g. PyTorch, Caffè...) tutte le reti sperimentate in questa fase sono state implementate tramite Keras con Tensorflow come backend. La prima rete provata è stata una convoluzionale *PSPnet 101*[96] (Pyramid Scene Parsing Network) pre-trainata sul dataset *Pascal voc12*[97] (la Visual Recognition Competition 2012 è una collezione di immagini creata per una challenge di Object recognition di scene realistiche bandita nel 2012 e ora molto usata per la training phase e test phase di

[95] https://towardsdatascience.com/semantic-image-segmentation

[96] https://medium.com/analytics-vidhya/semantic-segmentation-in-pspnet

[97] http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

algoritmi di image Segmentation). Questa rete ha un'architettura composta da tre blocchi di strati convoluzionali che costruiscono le feature map, a seguire un layer di Global Average Pooling che cattura i valori medi essenziali dell'output convoluzione e uno di interpolazione bilineare per riconvertire le immagini di input alla stessa dimensione (operazione di *Unsampling)*. Il risultato della PSPnet è una aggregazione delle feature map generate dai livelli convoluzionali e quelle fornite dall'Unsampling e dal averaging pooling. La seconda rete utilizzata invece è sempre una CNN chiamata U-Net e pre-allenata attraverso un set di dati di Human Parsing. La sua architettura, detta anche *Fully Convolutional Network*, è composta da due percorsi simmetrici (lato compatto e lato espanso) a forma di U, da qui il nome. I due percorsi hanno diversa architettura e funzione, il percorso compatto è una classica sequenza di rete CNN (conv layer, ReLU layer e max pooling) ed è utilizzata per mettere in risalto le caratteristiche principali delle immagini anche a costo di una riduzione della risoluzione. Il secondo percorso invece alterna un processo di Unsampling con quello di convoluzione, concatenazione con le feature map del lato ad esso simmetrico e linearizzazione tramite ReLU. Questi due percorsi si legano e vengono combinati in un ultimo strato di convoluzione che svolge il ruolo di assemblatore per la generazione di un output accurato. La terza ed ultima rete sperimentata invece è una DeepLab V3+ costruita da Google nel 2016 e pre-trainata sul dataset *Cityscapes* [98](collezione di immagini per la comprensione semantica delle scene urbane). La DeepLab V3 plus come la U-net è una rete con architettura encoder-decoder ma che adopera una convoluzione particolare chiamata Atrous (forma generalizzata di convoluzione con cui si è in grado di catturare e processare le informazioni multidimensionali). In sintesi, sono presenti due moduli: il primo di encoder svolge la codifica delle informazioni su più dimensioni utilizzando la Atrous Convolution, il secondo (decoder) decodifica la segmentazione generata evidenziando i bordi delle figure e degli oggetti.

Dopo questo breve excursus sulle architetture delle reti utilizzate, è possibile ora presentare al lettore la parte pratica di costruzione del secondo dataset e il processo di scelta della rete migliore per l'implementazione della semantic image Segmentation sul dataset RGB. Scelta che verrà effettuata confrontando le immagini di output delle diverse reti e fornendo qualche numerica sull'interpretabilità dei risultati per la VGG16. Lo script adoperato verrà esplicitato

[98] https://www.cityscapes-dataset.com/dataset-overview/

e descritto solo per la rete scelta, al fine di dare al lettore una visione più focalizzata sul trattamento dati effettuato senza creare eccessiva entropia nella spiegazione. Come mostrato in *Fig. 5.2*, osserviamo i risultati dei tre modelli di semantic image Segmentation.

Figura 5.2: Esempi di risultati di Image Segmentation su una mezza figura rappresentante R. Downey Jr in Tony Stark (tratta da "*Ironman*" di J. Favreau). A sx output PSPnet 101[99]. Al centro output U-Net[100]. A dx output DeepLab V3+[101]

Non è difficile intuire da questi output quale sia il risultato che meglio mette in risalto gli oggetti, in questo caso la figura umana, presenti all'interno dell'immagine e quindi permette di raggiungere l'obiettivo di semplificare i dati grazie alla segmentazione rendendoli di facile analisi per una CNN ai fini della classificazione. Questa differenza si riflette anche nell'interpretabilità della VGG16 rispetto a tali immagini di output, valutata tramite l'accuracy sul validation set ottenuto durante la training phase effettuata su un campione di settemila immagini output per ogni rete. Per tutte le reti, naturalmente, abbiamo ottenuto un'accuracy sul training set superiore al 90% con quaranta epoch ma per la PSPnet 101 l'accuratezza sul set di validazione è del 55% che è decisamente bassa ma prevedibile vista la poca delineazione delle figure all'interno delle immagini (molte immagini di inquadrature come il campo lungo risultato essere solo del colore di sfondo, senza altri oggetti all'interno). L'accuracy della U-Net invece, come il lettore può intuire

[99] https://github.com/divamgupta/image-segmentation-keras

[100] https://github.com/kairess/human_segmentation

[101] https://github.com/bonlime/keras-deeplab-v3-plus

data l'immagine d'esempio, è solo del 38%. Questo perché l'algoritmo, nonostante riconosce e abbia la capacità di delineare i bordi, non riesce a catturare l'intera figura senza alterarla. Questo invece non succede al risultato della rete DeepLab V3+ che cattura in modo efficiente la figura, delineando i bordi, segmentandola dal rumore di fondo e rendendola molto riconoscibile per la rete. Infatti, questo risultato si riflette sull'accuracy che la VGG16 riesce a raggiungere che, come vedremo nella presentazione dei risultati, supera il 70% (la rete interpreta in modo abbastanza efficiente le figure e riesce a discernere tra le diverse inquadrature con una buona accuratezza, già superiore ad altri approcci nello stesso campo ma naturalmente, come soluzione stand-alone, non potrebbe essere applicata non esistendo nel cinema immagini del genere).
Per concludere l'approfondimento su questo secondo dataset, presentiamo lo script con il quale queste immagini sono state generate (DeepLab V3+) e i relativi commenti.

```
import Numpy as np
from PIL import Image
from matplotlib import pyplot as plt
import os
import cv2

from deeplab import Deeplabv3
```

Nel preambolo dello script di creazione dell'Image Segmentation Dataset vengono importate le librerie principali utili al caricamento delle immagini di base dalla fonte e alla manipolazione degli array. A seguire è importato dall'omonima libreria il modello Deeplabv3 pre-trainato che sarà il cuore di questo algoritmo.

```
source_path="C:/Users/Francesco/Desktop/DatasetRGB/Mezzo Busto"
destination_path="C:/Users/Francesco/Desktop/DatasetSegmentation/Mezzo
Busto"

trnd_image_wdth=512
mean_subtrac_val=127.5
```

```
count=1
for image in os.listdir(source_path):
   try:
      image_path = os.path.join(source_path,image)
      img=np.array(Image.open(image_path))
```

Con questa sezione iniziale di script entriamo nel cuore dell'algoritmo di estrazione delle immagini segmentate. Come si può notare dai path introdotti, l'algoritmo è stato implementato classe per classe ma si potrebbe prevedere, inserendo un vettore di categorie e un ciclo for ulteriore, un algoritmo *all in one* che generi l'intero dataset in un run con tempistiche più elevate (l'algoritmo è stato previsto in questo modo perché, agendo classe per classe, è possibile valutare i risultati più volte all'interno del processo di creazione del set di dati senza attendere un'intera esecuzione del modello). Successivamente al settaggio dei percorsi, all'interno del ciclo for vengono lette le immagini (il path della cartella viene concatenato alle immagini al suo interno) utilizzando, diversamente degli algoritmi precedenti, la funzione open della libreria Image al posto di ImRead di OpenCV al fine di non dover prevedere la trasformazione da BGR a RGB (Image, diversamente da OpenCV legge le immagini direttamente in spazio colore RGB). Il resto dell'algoritmo, che vedremo all'interno dello stesso ciclo for, è utilizzato per generare per ogni immagine le etichette utili al processo di segmentazione che avviene tramite predizioni sui pixel effettuate dal modello.

```
wdth, hght, _ = img.shape
      ratio = float(trnd_image_wdth) / np.max([wdth, hght])

rszd_img=np.array(Image.fromarray(ing.astype('uint8')).resize((int(ratio*hght),
int(ratio * wdth))))

rszd_img = (rszd_img / mean_subtrac_val) - 1.
```

In questo frangente l'algoritmo ridimensiona l'immagine alla massima dimensione presente nel training set così da avere immagini tutte della stessa dimensione e abbastanza grandi da essere analizzate in modo efficiente dalla rete. A seguire viene

effettuata la normalizzazione dell'immagine di training ridimensionata così da ridurre il carico computazionale e da permettere una processing più semplice per il modello.

```
pad_x = int(trnd_image_wdth - rszd_img [0])
pad_y = int(trnd_image_wdth - rszd_img [1])
rszd_img = np.pad(rszd_img, ((0, pad_x), (0, pad_y), (0, 0)), mode='constant')

deeplab_mod = Deeplabv3()

result = deeplab_mod.predict(np.expand_dims(rszd_img, 0))
lbls = np.argmax(result.squeeze(), -1)
```

Qui l'immagine viene quadratizzata (*padding)* per ridurre i possibili problemi di memoria (*memory conflicts*) e conservare la coerenza di dimensioni tra questa e i dati in input, prima delle predizioni effettuate dal modello presenti nelle linee di codice successive. Quindi, caricata la rete DeepLabV3+ su una variabile, attraverso la funzione predict, viene creato un vettore di predizioni fatte sull'immagine ridimensionata e *"paddata"*. Da questo vettore sono estratte le etichette predette per l'immagine, fulcro del processo di image segmentation, con cui è costruita la stilizzazione del dato.

```
if pad_x > 0:
        lbls = lbls[:-pad_x]
    if pad_y > 0:
        lbls = lbls [:, :-pad_y]
    lbls = np.array(Image.fromarray(labels.astype('uint8')).resize((hght, wdth)))
```

A valle delle predizioni il padding viene rimosso e l'immagine è riportata alla dimensione originale prima di essere salvata sul path di destinazione attraverso la funzione *plt.savefig* di matplotlib. Il nome dell'immagine è costruito attraverso un contatore incrementale concatenato con il nome della classe di origine.

```
plt.imshow(lbls)
```

```
plt.savefig(os.path.join(destination_path,(str(count)+'MF.jpg')))
```

Come si può vedere le immagini segmentate sono contenute all'interno della variabile corrispondente ai labels di predizione.

5.1.3 *Dataset 3: Hypercolumns*

La parola Ipercolonne[102], come altre terminologie del Deep Learning, è comunemente usata in ambito neuroscientifico per chiamare un gruppo di neuroni di tipo V1 (appartenenti alla corteccia visiva primaria) con una spiccata sensibilità al riconoscimento dei bordi con orientamento e frequenza multiple e struttura a colonne, da qui il nome *Hypercolumns*. Le Ipercolonne quindi, sono una metodologia adoperata per ottimizzare le predizioni delle CNN e l'estrazione delle feature fondamentali di un'immagine. Le reti convoluzionali, solitamente, si avvalgono dell'ultimo layer fully connected per estrarre la rappresentazione delle feature dei dati in input ma sfortunatamente le informazioni generate in quello strato finale risultano essere spazialmente imprecise per permettere di effettuare una localizzazione efficiente delle caratteristiche a causa della catena di operazioni di pooling ecc. Allo stesso tempo, gli strati precedenti a quello fully connected forniscono informazioni di localizzazione molto precise ma con un gap in merito alla semantica. Come dimostrato nell'articolo del Dr. Hariharan (professore alla Cornell University) *"Hypercolumns for Object Segmentation and Fine-grained Localization"* [16], grazie all'estrazione delle Ipercolonne per ogni pixel, utilizzate poi come vettore di attivazione dell'insieme delle unità convoluzionali su ogni pixel, è possibile catturare le informazioni relative alla localizzazione spaziale dai layer intermedi, aggiungendo una semantizzazione derivante dal FC layer considerato come una feature map. Questa tecnica garantisce risultati, in fase di previsione, molto più performanti.
L'estrazione delle Ipercolonne [103]per un'immagine si divide in diverse fasi che ora andremo ad approfondire mostrando il codice che permette questa operazione.

[102] https://towardsdatascience.com/review-hypercolumn-instance-segmentation

[103] https://blog.christianperone.com/2016/01/convolutional-hypercolumns-in-python/

```
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16
import os
from pathlib import Path
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import seaborn as sns
sns.set_style("white")
import scipy.misc
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array,
load_img

vgg [104]= VGG16(weights='imagenet')
```

Per iniziare, come si vede dall'estratto di script, oltre all'import delle librerie si deve prevedere l'utilizzo di una rete convoluzionale da alimentare con le immagini per cui si vogliono estrarre le ipercolonne, al fine di ottenere le feature map per ogni location contenuta in esse. In questo codice è stata utilizzata una VGG16 già allenata sul dataset ImageNet e importata tramite Keras, essendo questa una rete molto performante per tale scopo grazie alla sua ottima accuratezza in classificazione di diverse tipologie di figure. Un problema in cui spesso si può incappare quando si svolge questa operazione è la differenza di dimensioni tra le feature map estratte (ridotte a causa dell'operazione di riduzione del dimensionamento generata dai layer di pooling) e le immagini di input che risultano essere più grandi. Questa issue è stata superata effettuando l'unsampling bilineare della mappa delle caratteristiche che quindi conserva la stessa dimensione dell'immagine di partenza. Un altro problema superato dagli autori del paper citato è quello derivante del passaggio della semantica delle informazioni dei layer precedenti a quelli Fully Connected. Non possedendo quest'ultimi tale

[104] https://www.kaggle.com/rupeshs/hypercolumn-test-vgg16

informazione e non essendo possibile catturare le unità semanticamente legate al pixel processato, è stata scelta come soluzione la creazione di una mappa delle attivazioni dei layer FC come una feature map 1x1. Ciò porta ad avere ogni posizione dell'immagine legata alla stessa informazione relativa ai livelli finali dell'ipercolonna e il raccordo di tali attivazioni ci fornisce le Ipercolonne desiderate.

```
source_path="C:/Users/Francesco/Desktop/DatasetRGB/Primo Piano"
destination_path="C:/Users/Francesco/Desktop/HypeDataset/Primo Piano"
count=0

for image in os.listdir(source_path):
    try:
        image_path=os.path.join(source_path, image)
        count+=1
        image = image.load_img(image_path, tg_size=(224, 224))
        image = image.image_to_array(image)
        image = np.expand_dims(image, axis=0)
        image = preprocess_input(image)

        predictions = vgg.predict(image)

        plt.plot(predictions.ravel())
```

Figura 5.3: Esempio di primo piano, rappresentante Joker (H. Ledger), contenuto nel dataset RGB (tratto da *"The Dark Knight"* di C. Nolan)

Nelle linee di codice appena inserite vengono lette le immagini (esempio di primo piano in Fig. 5.3) sempre con la stessa procedura descritta per gli altri algoritmi (ciclo for svolto per ogni classe di inquadrature, concatenazione del path di origine con le immagini al suo interno e infine apertura dell'immagine). Prima di fornire l'immagine alla rete per estrarre le predizioni e quindi le feature maps, questa viene manipolata, espandendo le dimensioni fissando un asse, quindi preprocessata tramite il metodo di Preprocessing delle VGG16 che rende le immagini adatte ad alimentare tale CNN. Infine, le immagini, una alla volta, sono date in input al metodo di predizione. Dalla Fig. 5.4 possiamo osservare le attivazioni finali dello strato softmax sulle classi della VGG16.

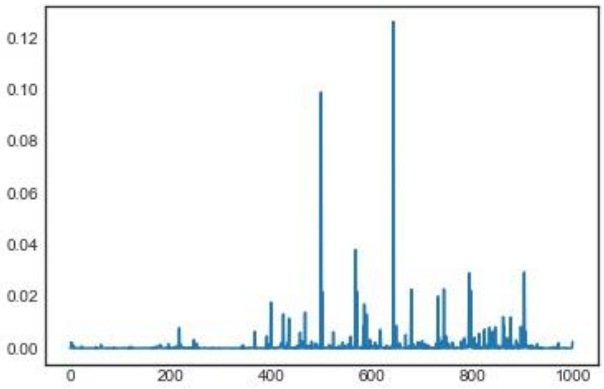


Figura 5.4: Performance in predizione della VGG16 su un esempio di primo piano

Successivamente si entra nel vivo della generazione delle Ipercolonne di un'immagine, provvedendo ad estrarre le attivazioni delle feature map che dipendono dal layer scelto nella sequenza di strati della rete. Per l'estrazione finale è stato scelto il secondo layer, strato convoluzionale, perché come si nota dalla Fig. 5.5 che mette a confronto l'attivazione della mappa delle caratteristiche del secondo livello con quella di uno dei livelli fully connected di output, le feature catturate dai livelli che terminano la rete, come già preannunciato, forniscono una visione grossolana , grezza e allungata dell'immagine (le feature map dei primi livelli possiedono una dimensione maggiore rispetto a quelle dei layer finali a causa delle operazioni di pooling. Ciò porta ad avere una perdita nella qualità dell'immagine che risulta stratchata), differentemente dallo strato convoluzionale scelto che riesce ad estrarre efficientemente le caratteristiche della figura all'interno del primo piano e a metterla in risalto.

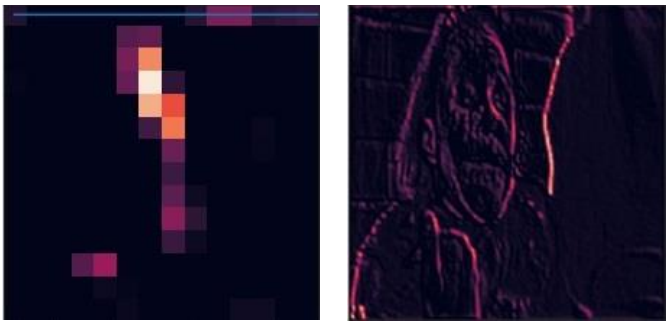


Figura 5.5: Esempio di attivazione della feature map generata da un layer FC di output (a sx) e da un livello convoluzionale iniziale (a dx)

Questa operazione è svolta dal codice sottostante che ottiene le feature del secondo layer tramite la funzione di Keras backend. Le caratteristiche estratte vengono poi manipolate e trasposte per farle tornare alla dimensione originale dell'immagine.

```
get_feat = K.function([vgg.layer[0].input], [vgg.layer[2].output])
ft = get_feat([x])[0]
print(ft.shape)
convolutional_out = ft[0, :, :, :]
ft_map = convolutional_out.transpose((2,0,1))
plt.imshow(ft_map[2])
```

Arrivati a questo punto si giunge al cuore pulsante dell'algoritmo e dell'estrazione delle Hypercolumns tramite la funzione *extract_hype*.

```
def extract_hype(model, layer_index, inst):
    layers = [model.layers[li].output for li in layer_index]
    get_feat = K.function([model.layers[0].input],layers)
    faet_maps = get_feat(inst)
    hypercols = []
    for convolutional_map in feat_maps:
        cv_out = convolutional_map[0, :, :, :]
        feat_map = cv_out.transpose((2,0,1))
        for feature_map in feat_map:
            upscaled =resize(fmap, (90, 160), mode='constant', preserve_range=True)
            hypercols.append(upscaled)
    return np.asarray(hypercols)
```

Tale funzione necessita di alcuni parametri per funzionare che saranno forniti all'interno del ciclo for (questa funzione è dichiarata al di fuori del ciclo): la CNN che si vuole utilizzare (in questo elaborato, come già accennato, utilizziamo la VGG16 ma il numero di modelli presenti in letteratura e utilizzabili è illimitato), un serie di indici di strato utili all'estrazione stessa delle Hypercolumns e, infine, un'istanza con cui, come vedremo, vengono passate le immagini alla funzione, una alla volta. All'interno della funzione le immagini ipercolonnari generate vengono anche scalate alla dimensione desiderata (160x90) così da essere già pronte per lo step successivo di allenamento in transfer learning di un'altra VGG16.

```
layers_to_extract = [1]
    hypecol = extract_hype(vgg, layers__to_ extract, [x])
    avg_hypecol=np.average(hypecol, axis=0)

    plt.imshow(avg_hypecol)
    plt.savefig(os.path.join(des_path,(str(count)+'PP.jpg')))
```

La fase finale di questo algoritmo, che ci porta quindi alla creazione del dataset di Hypercolumns desiderato, consta di un'ultima operazione, ovvero quella di alimentare la funzione con l'immagine da estrarre. Come anticipato, l'immagine ipercolonnare dipende dal layer da cui viene generata la feature map la quale varia in termini di qualità delle attivazione delle caratteristiche. In questo algoritmo abbiamo utilizzato il primo livello convoluzionale che ci permette di avere una figura con le caratteristiche ben delineate e i contorni messi in evidenza, effetto molto utile allo scopo dell'elaborato (Fig. 5.6 - sx). In caso si decida però di provare a modificare il layer adoperato per l'estrazione delle Ipercolonne (e.g. il quindicesimo come in Fig. 5.6 - dx), più il livello sarà vicino allo strato di output più l'informazione sarà spazialmente astratta e grezza.

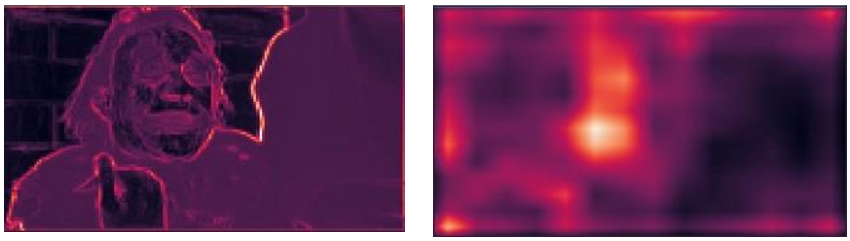


Figura 5.5: Esempio di immagine ipercolonnare estratta dal primo layer convoluzionale (a sx) e dal layer FC (a dx)

5.2 **Data Exploration e complessità del problema**

La *best practice* per qualsiasi problema correlato ai dati è sempre effettuare una prima analisi delle risorse che si hanno a disposizione e quindi un'esplorazione in merito alle caratteristiche dei dataset presi in considerazione, al fine di individuare le criticità che rendono la task da svolgere molto complessa.

A tal proposito è stata inserita questa sezione di *Data Exploration* a valle dell'approfondimento riguardante i dataset adoperati nella fase sperimentale di questo elaborato, così da mostrare al lettore quanto il problema affrontato sia complesso e quindi come i risultati che saranno presentati siano un importante primo traguardo verso un maggiore utilizzo del Deep Learning nel campo cinematografico. Questo viaggio nella complessità del problema parte dal campione di immagini che sono alla base della classificazione delle inquadrature e da cui deriva naturalmente la difficoltà, per molti algoritmi, di ottenere dei buoni risultati in merito allo smistamento in classi diverse. Per questo motivo, si è andati più in dettaglio nel set di dati portando avanti un analisi ad-hoc tramite alcuni algoritmi di *dimensionality reduction* e *clustering*, molto utili per fare un primo screening sulle correlazioni e le distribuzioni dei dati, fonti di complessità. Tale studio prende in considerazione solo il set di inquadrature RGB perché gli altri dataset hanno un valore dimostrativo minore, in relazione all' alta complessità del problema. Ciò è dovuto proprio alle trasformazioni effettuate che tendono a rendere il dato semplificato a livello di caratteristiche e quindi non rappresentativo per la definizione del problema originale. Gli algoritmi implementati per svolgere la data

exploration sono la Principle Component Analysis[105] (da ora PCA), il t-SNE[106] e il K-means clustering.

I primi due sono metodi di riduzione delle caratteristiche dei dati *high-dimensional*[107] molto utilizzati per la semplificazione del problema di base in modo tale da poterlo meglio studiare e approfondire tramite visualizzazione. Il terzo è un algoritmo di Unsupervised machine learning che ha la capacità di riconoscere pattern comuni tra i dati senza avere il bisogno di alcuna etichetta prestabilita e che, nel contempo, permette di effettuare una *visual analysis* molto intuitiva sulla correlazione tra i campioni adoperati. Quest'ultima tecnica di *correlation detection* è molto utile per comprendere la difficoltà di classificazione delle otto tipologie di inquadrature adoperate nel mondo del cinema (dati altamente dimensionali), per un algoritmo non addestrato appositamente a svolgere tale ruolo. Questa dimostrazione darà quindi più valore scientifico alla scelta di un approccio in modalità Supervised tramite metodologia di Deep Learning.

L'algoritmo di clustering in questione non è molto efficace su dati con un numero di feature troppo elevato ed è per questo che tale studio ha inizio dall'utilizzo degli algoritmi di riduzione della dimensionalità così che poi si possa svolgere un'analisi ottimale di clustering partendo da un set di dati con caratteristiche ridotte.

5.2.1 *Dimensionality reduction tramite Principle Component Analysis e t-SNE*

Prima di iniziare con lo studio descritto brevemente nell'introduzione è necessario fornire al lettore un rapido excursus teorico sugli strumenti qui adoperati e soprattutto sulle motivazioni di implementazione.

Le immagini costituenti gli scatti cinematografici, come già detto, sono dati molto complessi e N-dimensionali, caratterizzati da un numero elevatissimo di feature (43200 in questo elaborato ma solo dopo uno scaling massivo ad una risoluzione relativamente bassa come il 160x90) e ciò li rende di difficile interpretazione per

[105] https://towardsdatascience.com/k-means-and-pca-for-image-clustering-a-visual-analysis

[106] https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python

[107] Per *high dimensional* si intendono dati influenzati da molte variabili e che quindi possiedono un alto numero di dimensioni. Questa caratteristica li porta ad essere molto complessi e di difficile esplorazione

qualsiasi strumento di analisi visivo o statistico. Il problema fondamentale è che ogni algoritmo non riesce a processare nel modo corretto, focalizzandosi sulle feature più importanti, un set di dati così altamente dimensionale e per tal motivo è abitudine, per svolgere una data exploration, iniziare lo studio con metodologie di dimensionality reduction. Queste tendono, ognuna con tecniche matematiche o statistiche diverse, a ridurre il rumore e la variabilità dei dati e quindi l'alto numero di caratteristiche, garantendo la conservazione della maggior parte dell'informazione. Il risultato di questi algoritmi quindi porta a concentrarsi sulle sole feature rappresentative (informazioni principali) e ad analizzare i dati in modo visivo con la finalità di effettuare un primo studio su questi. I due metodi scelti sono la Principle Component Analysis e il t-SNE, entrambi facilmente implementati in python grazie alla libreria di machine learning *sklearn*. La PCA è una metodologia di statistica multivariata utilizzata per decrementare il numero di dimensioni caratterizzanti un set di dati ma conservando comunque le informazioni principali. Questa è definita come *"una trasformazione lineare delle variabili che proietta quelle originarie in un nuovo sistema cartesiano in cui la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la variabile nuova, seconda per dimensione della varianza, sul secondo asse e così via. La riduzione della complessità avviene limitandosi ad analizzare le principali, per varianza, tra le nuove variabili".*[108] . In pratica la tecnica del PCA ricerca la correlazione tra variabili e conserva solo un minimo di queste al fine di spiegare una percentuale di varianza scelta e conservare l'informazione sulla distribuzione originale dei dati. Tale tecnica è implementata tramite risoluzione matriciale degli autovalori e degli autovettori che, insieme alla matrice di covarianza, rendono possibile osservare la direzione lungo la quale si dirige la variazione dei dati e trovare il punto di massima varianza di un set di dati. Semplificando con un esempio, un campione di dati con 1000 feature viene semplificato grazie al PCA e ridotto ad un numero di caratteristiche n che dipende dalla varianza che si preferisce mantenere all'interno del dataset. Più è alta la varianza conservata, più quantità di informazioni vengono mantenute (informazioni di poco valore aggiunto, spesso non necessarie per l'esecuzione della task desiderata e che quindi potrebbero essere trascurate al fine di ridurre la dimensionalità dei dati e semplificare il problema)

[108] https://it.wikipedia.org/wiki/Analisi_delle_componenti_principali

Il t-SNE (t-Distributed Stochastic Neighbor Embedding) è un'ulteriore metodologia per permettere di diminuire la dimensionalità di un dataset al fine di renderlo di facile visualizzazione nonostante il gran numero di feature e permettere una *visual analysis*. Tale tecnica, differentemente dalla precedente, non si avvale di strumenti di alta matematica per svolgere il compito di *dimensionality reduction* ma adopera metodi statistici e probabilistici con i quali ricerca la minore divergenza tra due diverse distribuzioni di similarità a coppie (elementi ad alta dimensionalità in input e corrispondenti punti a bassa dimensionalità). In sintesi, quello che il t-SNE cerca di fare è rappresentare nel modo più efficiente possibile i dati originari utilizzando però meno dimensioni possibili. Questa operazione è molto complessa a livello computazionale per dati N-dimensional come i nostri e quindi è stato necessario prima prevedere l'uso del PCA che effettua già una scrematura dell'informazione e porta ad una importante riduzione delle feature, rendendo così più semplice il compito del t-SNE e successivamente anche quello del K-means in fase di clustering.

Entriamo più nell'analisi presentando step by step la data exploration svolta la quale parte dall'implementazione della Principle Component Analysis. In primo luogo, sono state sfruttate le proprietà dei Numpy array per trasformare la matrice delle immagini di inquadrature 4D (9279, 90, 160, 3) in un vettore a due dimensioni (9279, 43200) appiattendo tre features (dimensioni h e w e i canali dei colori) in una sola dimensione e ottenendo un *single array* con le componenti totali (43200) che saranno oggetto della *dimensionality* reduction tramite PCA. Per implementare questa tecnica bisogna scegliere o una varianza minima che si vuole conservare (più è alta la varianza maggiore sarà l'informazione mantenuta e maggiore resterà la dimensionalità) o altrimenti il numero di features che si preferisce avere da cui poi deriverà la percentuale di varianza da esse spiegata. In questo studio si è preferito optare per la prima scelta così da valutare con una riduzione della varianza quante caratteristiche raggiunge il dataset preso in considerazione e quindi quanto "rumore" (informazioni con poco valore) viene rimosso. Utilizzando la modalità *Try and Error* il valore di varianza scelto è stato l'85% il quale porta a passare da 43200 a 115 feature (115 caratteristiche rappresentano l'85% della variabilità all'interno del dataset e tutte le restanti solo il 15%). Come osservabile dalla Fig. 5.6, l'immagine che contiene solo le principali 115 feature rappresentanti l'85% della variabilità dei dati è sicuramente meno definita e particolareggiata ma ha

abbastanza caratteristiche per essere analizzata in modo efficiente e sicuramente più rapido da un algoritmo di Unsupervised learning.

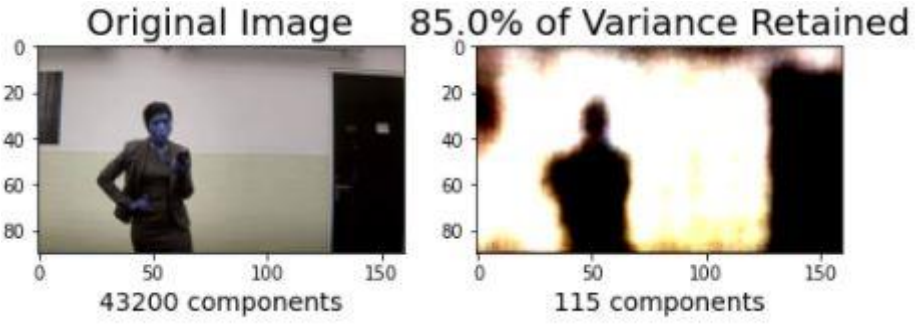


Figura 5.6: Esempio di un immagine contenuta nel dataset RGB prima e dopo la riduzione delle feature tramite PCA (varianza=0,85% n=115)

Questa notevole riduzione delle feature sarà molto utile per i successivi step di data exploration ma prima, date le 115 caratteristiche e l'85% di variabilità spiegata da esse, osserviamo attraverso un grafico 3D generato tramite libreria *plotly*, se l'utilizzo della PCA è già abbastanza per fare delle prime considerazioni sulla complessità del dataset provando a distinguere qualche pattern specifico. (Fig. 5.7)

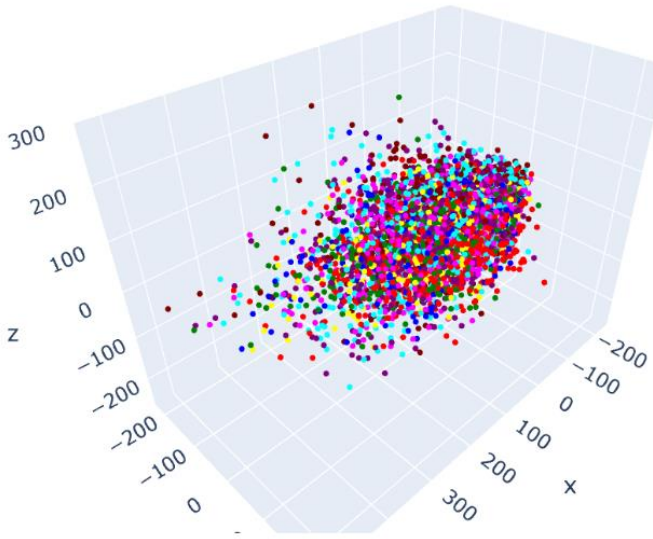


Figura 5.7: Rappresentazione grafica 3D tramite libreria plotly dell'output della PCA con n=115 composto da tipologie di inquadrature suddivise per colore

Quello che si nota dal grafico è un elevato rumore di fondo che caratterizza i dati e soprattutto una vicinanza molto accentuata tra le immagini di tutte le categorie.

Infatti, è semplice osservare che tutti i campioni sono molto ravvicinati tra loro nonostante la classe di appartenenza (*overlapping*) e concentrati in una piccola regione centrale con solo qualche outliers o dato sparso nel resto del piano. Questo ci fa dedurre quanto le inquadrature siano simili tra loro e quanto il problema di classificazione in modo accurato sia complesso. Da questo grafico è infatti impossibile distinguere le classi di inquadrature e la disposizione dei campioni di una categoria ha un andamento molto variabile sulle tre dimensioni. Il dataset quindi, nonostante la forte riduzione di feature, resta ancora molto complesso e, pertanto, di difficile analisi interpretativa. (naturalmente nel grafico è possibile osservare solo tre delle dimensioni caratterizzanti il dataset ma esse sono già sufficienti per valutare la complessità dei dati fino a questo step)

Giunti a questo punto, si può adoperare il secondo strumento introdotto in questa sezione, il t-SNE il quale è stato studiato appositamente per rappresentare graficamente su due dimensioni i dataset high dimensional. Al fine di implementare quest'altro algoritmo di dimensionality reduction sono state utilizzate le funzionalità dei dataframe della libreria di Pandas che permettono di semplificare la parte di plotting su due dimensioni (le matrice di immagini risultanti dalla procedura di PCA sono trasformate in dataframe con stesse dimensioni e feature: 9279,115). Il t-SNE, come la tecnica precedente, è stato implementato in Python grazie alla libreria di ML Scikit-Learn (tsne = TSNE (n_comp=2, verbose=1, perplex=40, iter=300). Quello che più interessa osservare di questa metodologia è la rappresentazione 2D del dataset la quale, grazie all'ulteriore riduzione di feature dopo quella svolta con la PCA, potrebbe mostrare nuovi pattern tra le immagini e quindi invalidare l'ipotesi finora fatta o confermare la presenza di molto rumore all'interno del dataset e l'overlapping dei dati.

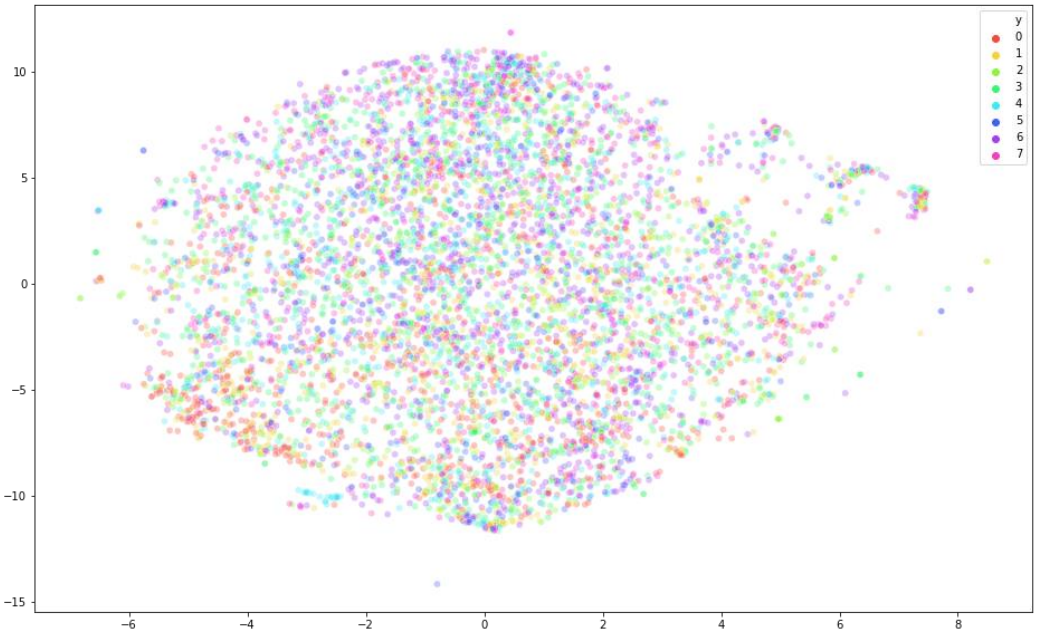


Figura 5.8: Rappresentazione grafica in 2D del set di dati *High dimensional* tramite t-SNE

Nel grafico abbiamo le immagini appartenenti alle diverse classi di inquadrature distinte da un colore diverso l'una dall'altra. Quello che si spererebbe di ottenere da questa tecnica sarebbe di riuscire ad intravedere una clusterizzazione/suddivisione in gruppi di dati appartenenti alla stessa categoria (con feature uguali o vicine) e quindi una serie di raggruppamenti dello stesso colore o quasi. La rappresentazione invece non lascia alcun dubbio su quanto il dataset sia pieno di rumore e il problema di classificazione complesso. Complessità dovuta sia al grande numero di feature, anche se qui ridotto, sia alla somiglianza tra una categoria e l'altra su tutte le caratteristiche. Infatti, dalla Fig. 5.8 non riusciamo ad osservare raggruppamenti di dati distinti ma solo una distribuzione molto variabile delle immagini appartenenti a tutte le categorie e un overlapping tra queste.

5.2.2 *Clusterizzazione tramite algoritmo K-means*

Uno step successivo alla dimensionality reduction, svolta tramite PCA e t-SNE i quali ci hanno mostrato visivamente come i dati adoperati siano caratterizzati da grande rumore da cui deriva poi la complessità del problema, nel percorso di data

exploration è implementare una clusterizzazione dei dati mediante l'algoritmo di Unsupervised machine learning, il K-means. L'obiettivo di questo studio è verificare se è possibile costruire dei cluster di qualità, rappresentanti le otto tipologie di inquadrature, anche solo con un processo di riconoscimento dei pattern tra dati o se, come si è già osservato nello studio appena effettuato, l'alta dimensionalità, l'elevato numero di caratteristiche e quindi la complessità del problema rendono difficile svolgere tale compito in modo accurato se non con una metodologia di Deep Learning. L'analisi quindi serve a dimostrare come l'unico approccio possibile per traguardare la classificazione degli scatti cinematografici con una buona accuracy sia l'uso di algoritmi di Supervised Machine Learning tra cui le Reti Neurali Convoluzionali e l'Ensemble Learning.

Prima di mostrare lo studio di data exploration effettuato, viene fornita al lettore una breve overview sul clustering e sulla tecnica del K-means con lo scopo di permettere una migliore comprensione di ciò che è stato implementato.

Il clustering è una metodologia di Unsupervised machine learning la quale permette di suddividere in macro-gruppi i dati in base alla vicinanza delle caratteristiche dei campioni (in un cluster ci saranno solo dati con feature simili). Questo processo di raggruppamento avviene tramite il riconoscimento di pattern specifici e comuni tra i dati e avviene senza alcun'etichetta che specifichi l'appartenenza dei campioni a precise categorie di classificazione (questa caratteristica lo rende diverso dalle metodologie di Supervised machine learning presenti in questo elaborato). Esistono diversi algoritmi che permettono la clusterizzazione dei dati tramite logiche differenti, il più usato-e scelto anche per questa analisi- è il K-means.

Il K-means clustering permette di raggruppare i dati in K cluster indicati a priori creando un numero K di centroidi. I centroidi solitamente sono campioni che la tecnica individua come rappresentativi a valle di un processo iterativo di ottimizzazione del clustering che può avvenire più volte nell'arco della clusterizzazione. Ogni data point è assegnato ad un determinato cluster in base alla vicinanza ad un centroide, il cluster con il centroide più vicino al campione processato è assegnato a quest'ultimo. La "vicinanza" di un campione ad un cluster è calcolata tramite la distanza Euclidea[109]quadrata la quale viene minimizzata dall'algoritmo al fine di trovare il centroide più prossimo al dato da clusterizzare.

[109] $d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

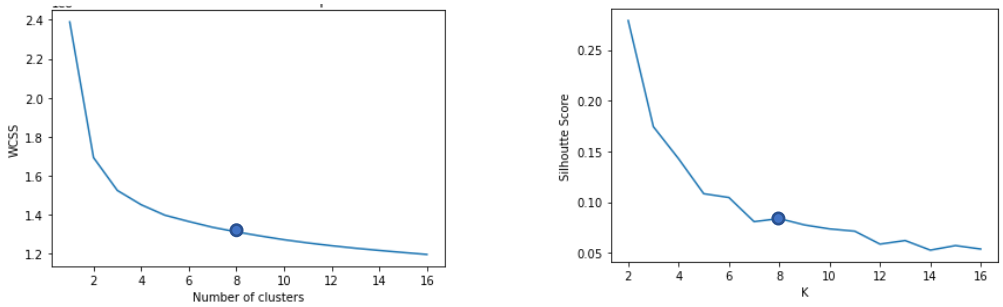
La prima cosa che bisogna analizzare per trarre le prime evidenze da questo studio è come individuare il numero K ottimale di cluster per generare dei raggruppamenti di qualità perché questo valore non si conosce a priori ma va comunque indicato tra i parametri dell'algoritmo. In realtà il numero di tipologie di inquadrature, come accennato nel capitolo quarto, è di otto classi nel campo del cinema e quindi quello che si vuole verificare è se per un algoritmo che distribuisce i dati in modo non supervisionato ma per pattern recognition, K=8 è il numero giusto per un clustering efficace. Ci sono diverse metodologie per trovare il valore ottimo [110]di centroidi da indicare all'algoritmo di K-means ma in questo studio ne sono stati implementati due: *Elbow Graph Method* e *Silhouette Method*. Il primo consiste nel calcolare la somma delle distanze dei campioni processati con il relativo centroide, anche detta Within-*Cluster-of Squared Error* [111](WSS), per un range da 0 a n valori di K. Il criterio di scelta del K ottimo è semplice e consiste nell'individuare quella porzione di grafico dove il WSS inizia a diminuire e quindi dove la curva si flette formando un gomito, da qui *Elbow Method*. Questo perché superando il gomito il guadagno in termini di minimizzazione del WSS è minore dello sforzo computazione che l'algoritmo deve affrontare nel suddividere i dati in un numero maggiore di cluster. Il *Silhouette Method* invece misura la similarità dei dati con il cluster a cui sono stati assegnati dall'algoritmo (calcolo chiamato coesione) e mette a confronto questo calcolo con lo stesso valore ma tra il punto e gli altri cluster. L'output di tale tecnica è uno score [112]con range (-1;+1) su n valori di K e il criterio di scelta è massimizzare il risultato del Silhouette Method prendendo il picco del grafico risultante. Questo indica il K in cui i dati sono stati assegnati nel modo più corretto. Dopo questa piccola introduzione si può continuare nell'analisi. Come già menzionato nel paragrafo sulla dimensionality reduction, l'algoritmo di K-means soffre l'alta dimensionalità di un dataset e con numeri elevati di feature potrebbe non riuscire ad effettuare una clusterizzazione sensata. Per tale motivo, al fine di facilitare il lavoro all'algoritmo, sia per la ricerca del K ottimo tramite le due metodologie che nell'implementazione vera e propria della clusterizzazione è stato

[110] https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means

[111] $WSS = \sum_{j \in n}(x_j - y_j)^2$ con centroide y_j e campione x_j https://medium.com/pursuitnotes/k-means-clustering-model-in-6-steps

[112] $f(x) = \begin{cases} s(j) = \frac{b(j)-a(j)}{\max[a(j),b(j)]}, & |C_j| > 1 \\ s(j) = 0, & |C_j| = 1 \end{cases}$

usato come dataset d'origine il risultante dal processo di Principle Component Analysis con $\sigma^2 = 0.85$ e n=115 feature.



a) b)

Figura 5.9: Elbow Method (a) vs Silhouette Method (b)

In Fig. 5.9 si nota che entrambe le tecniche adoperate in questo studio hanno un valore ottimo che non rispecchia il reale numero di classi di inquadrature presenti in letteratura. Il grafico di Elbow Method ci mostra una flessione tra i valori K=2 e k=4 che, in caso non si sappia quante siano le tipologie di scatti nel campo cinematografico, si potrebbero utilizzare come valori ottimi di clustering, invece in questo caso, conoscendo il desiderata relativo alle categorie di classificazione, viene dimostrata esclusivamente la difficoltà di un algoritmo di Unsupervised learning nel gestire tale problema di classificazione a causa della grande somiglianza tra le immagini delle diverse classi di inquadrature e l'overlapping tra queste già visto nei grafici precedenti. Il silhouette method è coerente con la tecnica precedente indicando i valori più alti di score tra i due e quattro cluster e quindi conferma l'ipotesi appena esposta in merito alla complessità di classificazione. Il lettore potrebbe chiedersi la motivazione per cui non si è deciso di utilizzare realmente questi ottimi trovati come numero di cluster: la risposta deriva dall'inapplicabilità nel campo pratico del cinema, di una classificazione a due o quattro inquadrature che inoltre risultano essere tentativi già sperimentati in letteratura e quindi non innovativi.

Il secondo passo predisposto per questa data exploration è stato eseguire un tentativo di implementazione di K-means clustering con K=8, desiderata della nostra classificazione. I parametri necessari all'esecuzione di tale algoritmo oltre al numero di cluster sono: la tecnica di inizializzazione dei centroidi (init) e il numero di volte che l'algoritmo eseguirà la clusterizzazione con centroidi diversi al fine di trovare il miglior risultato (n_init). Come init è stato scelto il più comune, il "*k*-

means ++", che permette la miglior scelta di inizializzazione e una convergenza rapida, il valore n_init=10 è stato scelto tramite procedura *Try* and *Error* valutando il risultato più efficiente. A valle del processo di clustering in otto classi operato dal K-means, possiamo analizzare i grafici risultanti su tre e due dimensioni:

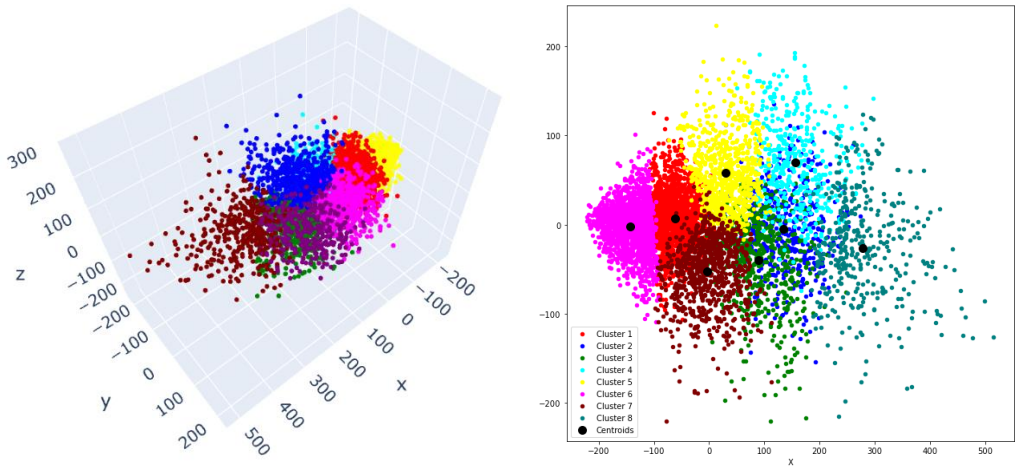


Figura 5.10: Rappresentazione 3D vs 2D del K-means clustering con K=8

La Fig.5.10 mostra come l'algoritmo di K-means riesca a generare 8 diverse classi ma anche come queste siano concentrate in una regione limitata del piano e completamente sovrapposte le une alle altre con notevole overlapping dei campioni di bordo. A confermare i grafici appena presentati, si è creato un set di *bar chart* i quali evidenziano la quantità di ogni inquadratura nei diversi otto cluster (Fig. 5.11)

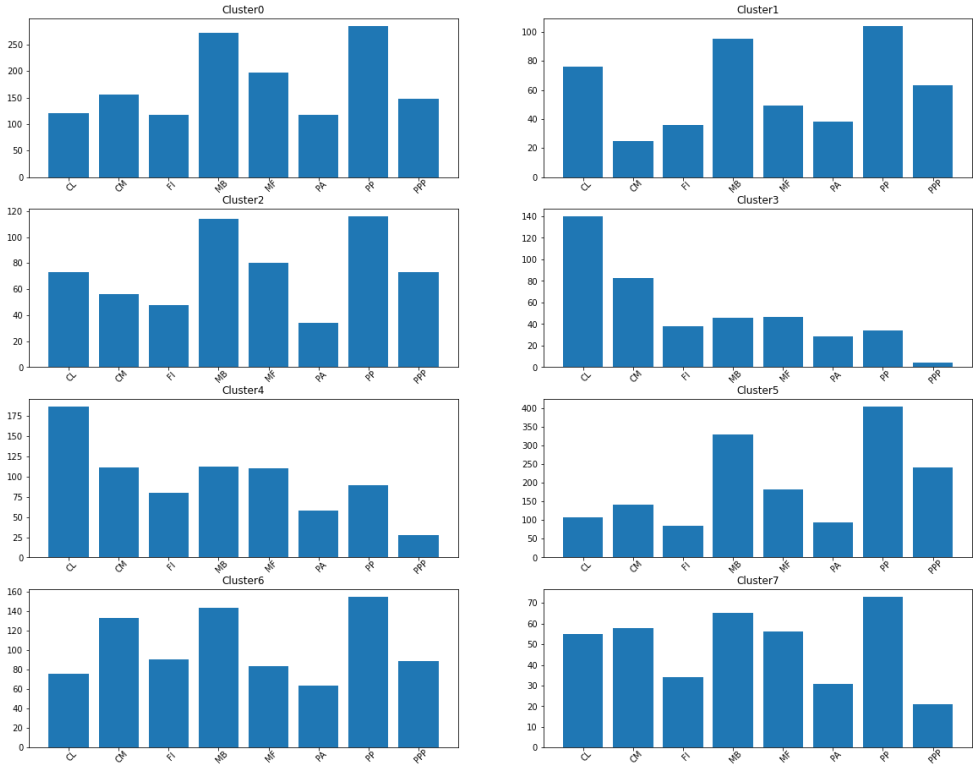


Figura 5.11: Visualizzazione su grafico a barre della quantità di campioni di un'inquadratura per ogni cluster generato

Il colpo d'occhio evidente che questa visualizzazione ci fornisce è la scarsa qualità dei cluster generati dal K-means, i quali contengono ognuno, in parti quasi uguali, tutte le tipologie di scatto cinematografico.

In conclusione, tutte le rappresentazioni presentante in questo studio sui dati, sia in fase di dimensionality reduction che in fase di clustering, confermano la complessità del problema di classificazione degli scatti in otto classi e soprattutto l'impossibilità, per un algoritmo non addestrato, di compiere questa determinata task, differentemente dal set di strumenti di deep learning adoperati in questa tesi che risultano invece in grado di adempiere a tali mansioni e svolgere una classificazione delle inquadrature cinematografiche abbastanza accurata da poter essere adoperata in campo pratico.

Queste evidenze confermano la bontà dell'approccio adoperato e la robustezza dei risultati ottenuti in fase di sperimentazione.

5.3 **Sperimentazione e classificazione delle inquadrature**

In questo paragrafo si entra nel vivo della fase di sperimentazione dell'elaborato di tesi che porterà, alla sua conclusione, all'esposizione di una metodologia di Deep Learning applicata alla classificazione delle inquadrature cinematografiche. All'interno di tale sezione verrà approfondito il modello scelto per raggiungere l'obiettivo di una classificazione accurata (Stacking Ensemble Learning di tre VGG16 pre-trainate e fine-tunizzate con tre dataset descritti nella sezione precedente) partendo dalla costruzione dei training set che andranno ad alimentare le tre reti VGG16, passando per il fine tuning dei layer finali delle stesse, utilizzando la collezione di dati appena create e finendo con lo script di creazione del modello finale di Stacking Learning e del training set adoperato dal quarto classificatore, cuore del modello, per migliorare le performance di classificazione. Il test del modello e i relativi risultati saranno invece esposti e commentati nel capitolo successivo insieme alla explanation della complessità dei dati della performance di classificazione della metodologia.

5.3.1 *Algoritmo di creazione dei training set per le VGG16*

Dopo la costruzione dei tre dataset, che saranno la base dell'intera sperimentazione e quindi della metodologia testata per effettuare la classificazione delle inquadrature cinematografiche, è necessario un altro passo fondamentale prima di poter allenare tramite transfer learning le tre reti VGG16 e quindi poi avere una buona accuracy grazie allo Stacking Ensemble Learning: la costruzione dei tre training set. Questa operazione è svolta tramite un altro script in Python che in breve legge le immagini presenti all'interno dei dataset creando e conservando traccia delle relative etichette. Lo script mostrato è solo per il training set generato dal dataset RGB ma, come il lettore può intuire, l'unica variazione da apportare per creare i restanti set di addestramento è la folder sorgente. Entrando nel dettaglio nell'algoritmo osserviamo il primo estratto:

```
import numpy as np
import tensorflow.keras
```

```
import os
import cv2
from tensorflow.keras.utils import to_categorical

WIDTH = 160
HEIGHT = 90

Dataset_RGB= "C:/Users/Francesco/Desktop/DatasetRGB/"
CATEGORIE= ["CL", "CM", "FI", "MB", "MF","PA","PP","PPP"]
```

Naturalmente come prima cosa sono importate le librerie necessarie a svolgere le operazioni di creazione del training set. A seguire, prima viene dichiarata la variabile Dataset_RGB a cui viene assegnato il percorso della cartella sorgente contenente tutte le immagini del primo dataset, e poi è inserita una variabile CATEGORIE che è un listato di sottocartelle presenti all'interno del Dataset_RGB e corrispondenti alle otto tipologie di inquadrature. In queste prime linee di codice è scelta anche la dimensione (altezza e larghezza) a cui saranno scalate le immagini prima di entrare nel training set. Come già descritto, prima di questa operazione le immagini hanno una risoluzione di 320x180 utile durante la fase di etichettatura per i motivi già esposti ma, successivamente, si rende necessario ridurre la risoluzione a 160x90 per permettere una minor carico computazionale per il calcolatore e quindi un processo più rapido.

```
Data_to_train = []
def create_training_data():
    for categoria in CATEGORIE:
        percorso= os.path.join(Dataset_RGB, categoria)
        num_classe= CATEGORIE.index(categoria)
        for image in os.listdir(percorso):
            try:
                image_array = cv2.imread(os.path.join(percorso,image))
                new_image_array = cv2.resize(image_array,(WIDTH,  HEIGHT))

                Data_to_train.append([new_image_array, num_classe])
```

```
            except Exception as e:
                pass
create_training_data()
```

La funzione in Python è il cuore dell'algoritmo di creazione del set di dati. In primis, tramite un doppio ciclo for sulle categorie e sulle immagini presenti all'interno delle sottocartelle, vengono letti tutti i dati presenti nel DatasetRGB (l'algoritmo prima apre e lavora tutte le immagini di una categoria creando un path per ogni immagine e poi passa alla successiva sottocartella). Successivamente sono estratte le etichette in forma numerica, corrispondenti alla cartella di appartenenza del dato, per ogni immagine tramite la funzione di indice (e.g. Primissimo Piano corrisponde all'etichetta 7, primo piano alla 6 ecc. naturalmente l'indice delle label parte da zero). A questo punto lo script manipola e scala un'immagine alla volta trasformando la matrice immagine in un vettore e in seguito ridimensionando quest'ultimo alla risoluzione scelta. Alla conclusione della funzione, l'immagine-array scalata e il suo label sono inseriti tramite metodo di append come lista a due posizioni (posizione 0 l'immagine e posizione 1 il label che lo caratterizza) nella lista vuota precedentemente dichiarata, Data_to_train. Questa operazione finale è effettuata per conservare la corrispondenza tra immagine ed etichetta prima di alcune operazioni che ora andremo a descrivere. Alla fine dello script queste due informazioni saranno separate così da avere un training set di immagini e una lista di label.

```
import random
random.shuffle(data_to_train)
X=[]
y=[]
for features, label in data_to_train:
X.append(features)
   y.append(label)
y=np.array(y)
y=to_categorical(y)
X=np.array(X)
```

Le righe di codice appena inserite hanno un ruolo molto importante all'interno dell'algoritmo. Per iniziare, i dati appena collezionati nella funzione precedente subiscono una randomizzazione tramite la funzione shuffle della libreria random. Questo è necessario per effettuare un training più efficiente che non porti a risultati inconsistenti o pessimi perché allenare una rete con una lunga sequenza di dati simili per poi passare ad un'altra con tipologia differente dalle prima e così via, (insieme di tutte le immagini di campo lungo e poi tutti i primi piani) crea dei problemi in fase di allenamento nel quale la rete, dopo la prima sequenza, credendo di aver imparato come si classifica (tutte le immagini sono campo lungo) non riesce più nel suo intento non riconoscendo i dati in input (sbaglia la classificazione di tutti i primi piani). In sostanza la rete perde l'abilità di generalizzazione dell'apprendimento e genera risultati con accuratezza molto carente. Il mescolamento dei dati viene effettuato in questo step per conservare, come già detto, la corrispondenza immagine-label. Successivamente alla randomizzazione, attraverso un ciclo for con doppio indice, la lista data_to_training viene divisa in due nuove liste, una per le immagini e una per le etichette che, anche se divise, mantengono sempre lo stesso ordine e quindi restano coerenti. Per convenzione, molto usata dagli addetti ai lavori nel campo machine learning, la lista con le feature (le immagini) è chiamata con una X maiuscola e invece quella contenente le etichette corrisponde ad una y minuscola. Per concludere, in questo estratto di script le etichette e le immagini vengono trasformate in Numpy Array perché, come vedremo successivamente, il metodo di allenamento della rete fornito da Tensorflow.Keras ha il limite di accettare solo array della libreria di Numpy. Altro passo finale, prettamente di forma, è la conversione delle etichette in un format più agevole per gli algoritmi di Deep Learning, la *One-Hot Encoding*. Tale codifica trasforma i label, attualmente rappresentati da una lista da 0 a 7 in cui ogni numero è una classe di inquadratura, in una lista di array ad otto posizioni tutte popolate dallo zero tranne una dove sarà presente l'uno. Le posizioni naturalmente corrispondono alle diverse classi e l'uno è inserito solo nella posizione relativa alla classe che corrisponde all'immagine (Campo Lungo= (1,0,0,0,0,0,0,0)). Questa codifica in questione risolve alcuni problemi in merito a casistiche di classificazione errata.

```
import pickle
```

```
pickle_out = open("training_images_RGB.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("training_labels_RGB.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Per finire, le due liste sono salvate in file formato Pickle (uno per le immagini e uno per i labels, conservando l'ordine corretto) al fine di avere sempre a portata di mano il training set senza dover ripetere l'esecuzione dell'algoritmo che, per grossi campioni di dati, richiederebbe anche un giorno di esecuzione. I file pickle possono sempre essere recuperati, aperti e salvati in variabili di un codice Python ed è proprio questo che effettueremo, come vedremo nel paragrafo successivo, per fornire i dati di allenamento alle nostre VGG16.

5.3.2 *Utilizzo della rete convoluzionale VGG16 e del transfer learning*

Dopo le spiegazioni fornite a proposito della costruzione dei dataset, del Preprocessing richiesto e della creazione dei training set essenziali per lo start della sperimentazione, possiamo entrare nella fase di implementazione della metodologia di Deep Learning e quindi dell'edificazione del modello in essere partendo dalla base, l'addestramento in transfer learning tramite i nostri dataset, delle tre VGG16. Prima però di approfondire l'architettura che contraddistingue l'algoritmo e quindi il codice vero e proprio di implementazione, è necessario fornire al lettore un breve excursus sulla struttura della rete convoluzionale VGG16 al fine di permettere una migliore comprensione in merito all'utilizzo e alle funzionalità di tale rete, riprendendo anche alcuni concetti già spiegati nei capitoli relativi alle CNN e al transfer learning.

Come già introdotto nella prefazione di questo capitolo, la VGG16 è una rete neurale convoluzionale chiamata anche OxfordNet dal luogo di provenienza dei ricercatori che l'hanno introdotta. É un modello che deve la sua notorietà tra gli

addetti ai lavori alla elevata accuratezza raggiunta durante i test su un dataset come ImageNet (collezione di milioni di immagini di dimensione quadrata 256x256 prese da internet e suddivise in migliaia di classi diverse). Importante è l'architettura che la caratterizza che ha permesso di raggiungere risultati notevoli. In Fig. 5.12 è mostrata una rappresentazione della struttura di una rete VGG16 e quindi la sequenza dei layer che la contraddistinguono.

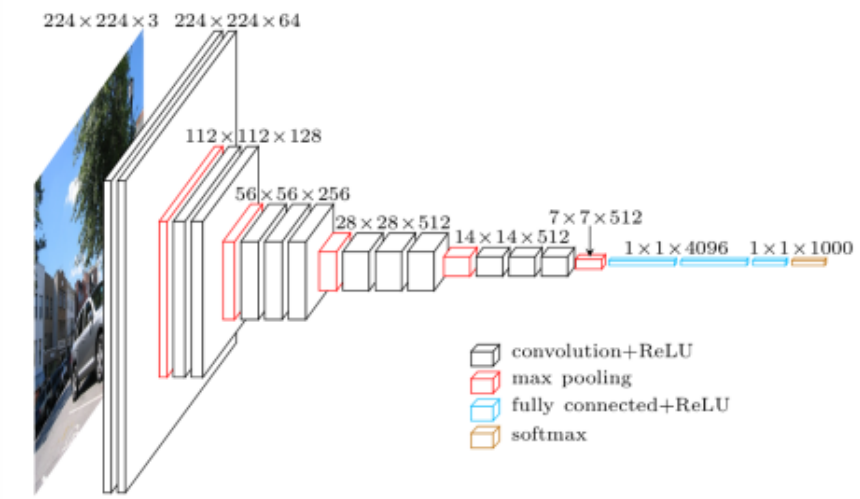


Figura 5.12[113]**:** Architettura generale di una rete VGG16 addestrata su ImageNet

Come da figura, osserviamo che l'immagine in input deve avere una dimensione 224x224 a colori su tre canali RGB così da poter alimentare il primo livello convoluzionale che sarà di 150.528 nodi (nella nostra VGG16 [114]fine tuned, l'input sarà modificato per permettere l'alimentazione della rete con le nostre immagini 160x90 e quindi ciò porterà alla riduzione del numero di nodi necessari). In realtà i layer convoluzionali iniziali, ma anche i successivi, sono raggruppati in blocchi convoluzionali. Per quanto riguarda il primo blocco, esso è caratterizzato da due layer a 64 canali, un filtro e un padding di dimensione 3x3 (il filtro è l'elemento principale che svolge la convoluzione e il padding invece permette di delineare i bordi di un'immagine in modo tale da non perderli durante le varie operazioni convoluzionali e per far sì che il filtro non vada fuori dall'immagine date le sue dimensioni matriciali). Inoltre, la dimensione 3x3 per il filtro risulta essere la minore per svolgere il compito di catturare l'informazione sulle quattro direzioni

[113] https://towardsdatascience.com/step-by-step-vgg16

[114] https://neurohive.io/en/popular-networks/vgg16/

(sopra, sotto, destra, sinistra). Continuando nella sequenza dei livelli presenti in una VGG16[115], troviamo subito un livello ReLU e uno strato di max pooling che seguono il blocco convoluzionale. Il primo layer serve ad eliminare i valori nulli generati dalla convoluzione e ad introdurre della non linearità in output lineare derivante da un calcolo lineare (prodotto scalare). Il layer di pooling invece è caratterizzato da un passo di azione sull'immagine 2x2 ed è utilizzato per ridurre la dimensionalità del problema, catturando le informazioni principali dell'input e migliorando le prestazioni in classificazione anche per immagini complesse. Questa sequenza descritta si ripete cinque volte creando cinque blocchi convoluzionali 3x3 (formati da due o da tre layer convoluzionali) seguiti sempre da un livello ReLU e da uno di max Pooling 2x2. Ciò che cambia tra i vari livelli è solo la dimensionalità dell'output (la feature map) che si riduce e diventa meno lineare grazie alle operazioni descritte (da 224x224 a 7x7 all'ultimo max pooling) e il numero di filtri/canali che sale da 64 fino a 512. A valle di questa sequenza sono presenti i livelli Fully connected incaricati di svolgere la classificazione. Questa parte della rete cambia, in merito a numero di livelli e nodi, da architettura ad architettura e dipende dal problema da risolvere. In una struttura generale adattata alla classificazione di ImageNet, gli strati FC sono tre. Il primo appiattisce l'output matriciale multidimensionale fornito dal blocco conv-ReLU-pooling, trasformandolo in un array monodimensionale al fine di permettere un miglior lavoro di classificazione per questi ultimi strati completamente connessi. Il secondo è un dense layer con 4096 nodi come anche il flatten layer precedente ed è il livello che manipola i dati e li rende classificabili. Il successivo layer FC di output, di cui i parametri sono strettamente correlati al tipo di problema e al dataset da classificare, è un livello *Softmax* che si occupa della classificazione. Per il problema generale su ImageNet i nodi che caratterizzano questo layer sono mille, uno per ogni categoria da classificare, ma come vedremo a breve, attraverso il transfer learning, questi livelli di output saranno eliminati e ne saranno aggiunti altri con caratteristiche diverse da riallenare al fine di raggiungere l'obiettivo di suddividere i nostri dati in otto classi di inquadrature. Proprio per quanto riguarda la tecnica di transfer learning, che, ricapitolando, è la metodologia che sfrutta la conoscenza pregressa di reti pre-addestrate ri-allenando solo il layer finali su nuovi set di dati e congelando gli altri al fine di conservare i loro valori sinaptici, è stata utilizzata per

[115] https://www.geeksforgeeks.org/vgg-16-cnn-model/

costruire le nostre VGG16 in modo che svolgano la task di classificare in modo efficiente gli scatti cinematografici. Generalmente gli unici layer ri-addestrati sono i livelli di output FC che sono gli artefici della classificazione ma in questa sperimentazione si è deciso di ri-trainare, oltre che i livelli Fully Connected (eliminati e raggiunti con un numero di nodi minori partendo da 128 del Dense layer fino ad 8 nodi del Softmax, corrispondenti alle categorie di classificazione), anche l'ultimo blocco convoluzionale, ReLU e max pooling, così da migliorare l'accuratezza nella classificazione focalizzando l'apprendimento sulle tipologie di immagini di interesse e eliminando "lezioni" apprese dalla rete su categorie e feature poco rilevanti (le VGG16 sono allenate inizialmente su un dataset da categorizzare in 1000 classi di immagini ma, essendo la nostra classificazione solo di 8, sicuramente si può fare a meno di molte feature apprese dalla rete; inoltre, riallenare anche il blocco convoluzionale porta la rete a inserire nel proprio bagaglio di feature maps anche caratteristiche mai viste dalla rete finora e molto rilevanti per analizzare le nuove immagini di inquadrature provenienti da dataset manipolati con operazioni di pre-processing come ad esempio l'estrazione delle Ipercolonne).
Dopo questo excursus sull'architettura di una VGG16, generale o specifica per la task svolta in questa tesi, e sulla tecnica di transfer learning adoperata, è ora possibile mostrare al lettore il codice con cui è stato costruito questo primo modello di classificazione. Naturalmente questo codice è stato ripetuto per tutte e tre le reti convoluzionali allenate con i tre diversi dataset costruiti ma per semplicità al lettore verrà mostrato solo lo script di una delle reti implementate.

```
import pickle
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential, model_from_json, load_model
from tensorflow.keras.layers import Dense, Activation, Flatten,
BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Come sempre l'algoritmo inizia con l'import delle varie librerie in cui sono contenuti i metodi utilizzati. Una menzione speciale va alla libreria Keras con backend Tensorflow che permette di creare e allenare un modello di Deep Learning, operazione molto complessa, in un numero contenuto di righe di codice. Altra libreria di Machine Learning molto utile che ci darà una mano nella comprensione dei risultati di addestramento, è la Sklearn con la quale verrà implementata una confusion matrix dell'output al fine di osservare graficamente le performance della rete.

```
def Create_model():

vgg_16_model=VGG16(weights='imagenet',include_top=False, input_shape=(90, 160,3))

   for layer in vgg_16_model.layers[:-15]:
      layer.trainable=False

  model=Sequential()
  model.add(vgg_16_model)
  model.add(Flatten())
  model.add(Dense(128,activation='relu',kernel_regularizer=regularizers.l2(0.01)))
  model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
  model.add(Dense(8, activation='softmax')
return model
```

Il fulcro dell'algoritmo di costruzione e allenamento tramite transfer learning di una VGG16 è la funzione appena mostrata. Al suo interno viene per prima cosa scaricato un modello di VGG16 allenata su ImageNet con la variazione, già prevista dalla libreria Keras da cui è facilmente estratto, di alcuni parametri come l'input

shape che da 224x224x3 è adattata alle nostre immagini 16:9 e risoluzione 160x90x3 (lunghezza e altezza in Keras sono invertiti per convenzione, ecco il motivo per cui sono inserite all'inverso) e che quindi porterà ad un layer convoluzionale di input di 43.200 nodi. Altra variante rispetto ad un'istanza generale di VGG16 è la variabile booleana *include_top* impostata su *false* che permette di implementare una vgg già senza i livelli di output che andranno poi raggiunti con le caratteristiche utili allo svolgimento della task di interesse.

A seguire, si procede con il primo passo del transfer learning (o meglio del fine tuning della rete). Tramite un semplice loop che cicla su tutti i layer della VGG16 fino all'ultimo blocco convoluzionale non compreso e l'impostazione del booleano *layer.trainable=False*, si effettua il congelamento dei parametri (i pesi e il bias) dei livelli considerati, i quali non potranno essere ri-addestrati e conserveranno l'apprendimento pregresso. Come già introdotto, l'ultimo blocco convoluzionale sarà riaddestrato e per questo motivo è escluso da questa operazione (il ciclo for comprende solo i primi 15 livelli e quindi non l'ultimo blocco). A questo punto è arrivato il momento di costruire la nuova VGG16 che sarà lo strumento di classificazione dei nostri scatti cinematografici. Per semplicità, in questo caso è utilizzato un modello Keras di tipo "*Sequential"* ovvero con i livelli feedforward ma sarebbe possibile adoperare anche la tipologia *"Model"* che ha una gamma di funzionalità in più rispetto al modello sequenziale come la possibilità di collegare livelli non consecutivi, proprietà però non utile ai fini dell'elaborato. Creata un'istanza di modello sequenziale, tramite il metodo *Add*, l'intera VGG16 modificata ad hoc e con i layer congelati è inserita all'interno del nuovo modello feedforward. Questa rete neurale convoluzionale appena costruita però ha ancora un gap architetturale relativo ai layer di output e quindi non è ancora pronta al compito ad essa affidato, ovvero la classificazione. Per questo motivo sono create delle istanze di layer che poi vengono aggiunte una alla volta in coda alla nostra VGG16 contenuta nel modello sequenziale. Per prima cosa è inserito un livello *Flatten*, che come già discusso, trasforma l'output multidimensionale del *Conv-Block* in un array monodimensionale più semplice da gestire per gli strati fully connected in fase di predizione riducendo il carico computazionale. Il Flatten layer eredita il numero di nodi dalle dimensioni di ciò che riceve in input (un po' come il livello di input) e per questa motivazione non è dichiarato esplicitamente come negli altri strati. Il secondo e il terzo layer sono invece di tipo Dense che spesso

viene inserito in fase di classificazione perché implementa l'operazione di attivazione dei neuroni. Il numero di nodi dei Dense layer sono decrescenti al fine di donare una forma "a punta" alla rete che le permette di essere più stabile in vista dell'ultimo livello, per la precisione i nodi stabiliti sono 128 per il primo e 64 per il secondo. Altro parametro inserito all'interno di questi strati al fine di migliorare la stabilità delle operazioni svolte da questi è il Kernel_regulaizers[116]. Questi regolatori hanno la proprietà di ridurre il possibile Overfitting (problema in cui la rete performa bene sui dati di training ma in modo pessimo su altri dati, non riuscendo a generalizzare la lezione appresa) migliorando sia l'accuratezza sul training set sia sui dati di test. Ciò avviene con una somma delle penalità, impostate tramite parametro del regolarizzatore, e dei valori di funzione di perdita ottimizzata dalla rete (operazione svolta per ogni layer per cui è inserito il regulaizer). Ultima nota importante che contraddistingue i regolatori è l'esistenza di diverse istanze (L1, L2 e L1L2) le quali dipendono dal tipo di operazione che le penalità dichiarate svolgono in combinazione con i pesi del livello. In questo elaborato, per i dense layer, è stata usata la tipologia L2 che effettua una somma dei pesi al quadrato per ottimizzare la funzione di perdita.

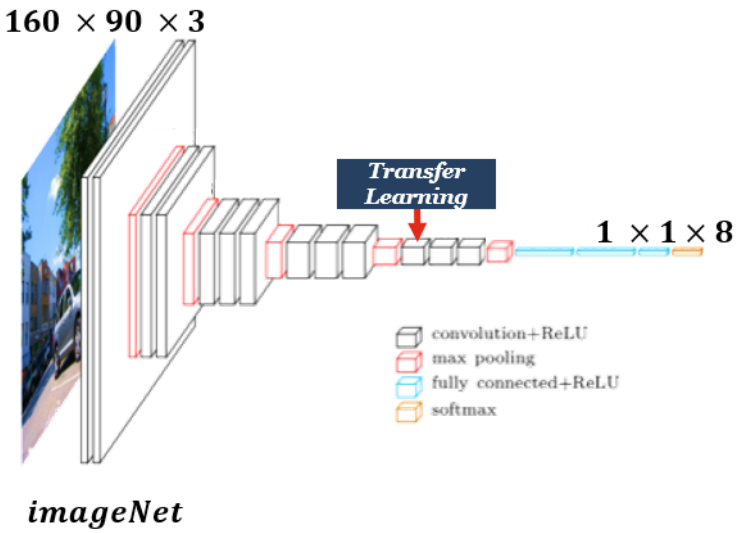


Figura 5.13: Architettura del modello creato tramite Transfer Learning

Tornando ai Dense layers inseriti, tutti sono caratterizzati da una funzione di attivazione ReLU, che elimina i valori nulli provenienti dalla convoluzione, tranne

[116] https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/

il livello di output, anch'esso uno strato Dense con 8 nodi (1 per ogni classe), che possiede una funzione di attivazione softmax, molto utilizzata nei problemi di classificazione multi classe, che assegna una probabilità ad ogni classe in base ai dati ricevuti dai livelli precedenti e così effettua l'assegnazione della categoria più corretta.

```
X=pickle.load(open("training_images_RGB.pickle","rb"))
y=pickle.load(open("training_images_RGB.pickle","rb"))

X=X/255.0
```

In queste poche righe viene effettuato quello che era già stato anticipato nel paragrafo di creazione del training set. Grazie ai file pickle precedentemente salvati è possibile ricaricare le immagini per l'addestramento e i relativi label salvandoli in due variabili globali del nuovo script. A seguito del caricamento è necessario prevedere un'azione di normalizzazione delle immagini contenute nel training set. Queste sono matrici multidimensionali formate da valori da 0 a 255 (valore del pixel) che, tenute a questo stadio, rendono il processo di allenamento della rete molto lento e *time consuming*. Esistono diversi metodi raffinati di *scaling* delle immagini ma per semplicità, lavorando con Numpy array che supportano le operazioni sulle celle delle matrici, la normalizzazione è effettuata in modo semplice dividendo la variabile matrice per il valore massimo possibile di una cella, 255. In questo modo la matrice immagine sarà ora composta da valori in un range da 0 a 1 senza grande perdita di informazione.

```
model=Create_model()
model.compile(loss="categorical_crossentropy",optimizer="sgd",
metrics=['accuracy'])
model.fit(X,y, batch_size=10, epochs=40, validation_split=0.04)
```

A questo punto si arriva al clou dello script. Nelle righe di codice inserite si richiama la funzione *Create_model ()* precedentemente descritta e quindi la VGG16 manipolata ad hoc è caricata su una variabile globale. Per effettuare l'addestramento restano solo tre step corrispondenti alla successive righe di codice che risultano essere quelle computazionalmente più gravose per il calcolatore

perché è lì che avvengono tutte le operazioni di aggiustamento pesi, Backpropagation, calcolo dell'anti gradiente stocastico etc, descritte nei primi capitoli. Tornando allo script, con il metodo *compile*, il modello viene compilato dal calcolatore e vengono qui dichiarate come parametri: la funzione di perdita/costo (qui è scelta la *categorical_crossentropy* ottimale per i problemi multi classe) e la funzione di ottimizzazione euristica (è stata utilizzata la *Stochastic Gradient Descent* perché molto efficiente ma esisto molti altri ottimizzatori come *Adam Optimizer*) e la metrica di performance con cui si vogliono visualizzare i risultati di training ad ogni epoch. Inserendo *accuracy* il calcolatore ci mostrerà l'accuratezza di predizione e il valore della perdita calcolato tramite Loss Function scelta. Come ultimi passi dell'addestramento vero e proprio restano solo due operazioni. La prima è impostare i parametri di training tramite la funzione *fit* che è il vero trigger dell'allenamento. A questo metodo vengono dati in input le immagini di training e i relativi label corretti così che il modello sappia il campione su cui allenarsi e sappia fare un confronto tra le sue predizioni e le etichette corrette (accuratezza), il batch_size invece rappresenta il numero di immagini che vengono processate contemporaneamente dalla rete (questo parametro è molto importante e va impostato con cura in base al numero di campioni che si hanno nel training set che nel caso in esame, non essendo molto elevato, è dichiarato a 10). Le epochs, come già descritto nel capitolo delle CNN, consistono nel numero di volte che il training verrà effettuato su quei dati (effettivamente è il numero di giri delle immagini nella rete e aumentandoli le performance dovrebbero migliorare ma con un maggiore rischio di Overfitting, per questo motivo qui è stato scelto un numero ragionato di 40 epochs). L'ultimo parametro ma non per importanza, ci fornisce l'unico e vero risultato preso in considerazione, che verrà mostrato e analizzato nel successivo capitolo, ovvero l'accuratezza e la perdita che il modello ha su un campione di dati non presenti tra quelli di training, il validation_set (performance di predizione del modello su un set di validazione). Attraverso il parametro *validation_split* del metodo *fit* è possibile creare un campione di validazione estraendolo in percentuale dal training set, questo non verrà più utilizzato per la fase di allenamento ma solo per una valutazione dell'accuratezza alla fine di ogni epoch. Naturalmente lo stesso vale per le etichette correlate. In questa sperimentazione viene usato l'4% del set di addestramento (formato in totale da 9279 immagini) che consta quindi di 372 immagini, campione ottimale per valutare le performance in

predizione della rete. Il motivo per il quale questo risulta essere l'unica metrica affidabile è che l'accuratezza sul training set può essere falsata dalla rete che tende a memorizzare i dati e la loro classificazione ma non impara le dinamiche e le regole con cui classificare altri dati (Overfitting). Essendo questo KPI calcolato su un set di dati mai visti dalla rete, lo rende un misuratore di performance molto affidabile.

```
model_json = model.to_json()

with open("Norm_X_StkLrn.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights('Norm_X_StkLrn_weights.h5')
```

A conclusione della fase di addestramento, operazione che richiede diverse ore nonostante sia solo degli ultimi layer tramite transfer learning e il set di dati non sia eccessivo, al fine di non ripetere il procedimento, l'architettura del modello viene salvata su un file JSON[117] così da poter essere sempre recuperata e riutilizzata in altri script e lo stesso vale per i pesi, calcolati in modo raffinato in fase di training, salvati in un file h5 (formato molto perfomante per questo tipo di informazioni). Come vedremo, il salvataggio di queste informazioni è fondamentale per riutilizzare le tre VGG16 durante la combinazione e il miglioramento dei risultati operato tramite Stacking Ensemble Learning. Infatti, proprio nell'implementazione di tale algoritmo, il primo passo sarà ricaricare questi modelli ed effettuare le predizione su tre nuovi set di dati (la costruzione, molto simile a quella del training set della VGG16, sarà approfondita nel successivo paragrafo), uno per modello, che saranno l'input di un quarto classificatore.

5.3.3 *Utilizzo della tecnica di Stacking Ensemble Learning*

[117] Il file JSON è un formato testuale di aggregazioni di dati come stringhe, numeri ecc in modo che siano facilmente recuperabili e riutilizzabili. I dati sono strutturati come un dizionario in cui in prima posizione troviamo la chiave e in seconda posizione, divisa dai due punti, troviamo il valore. https://devacademy.it/json/

In questa ultima sezione del capitolo 5 sarà presentata la metodologia innovativa di classificazione delle inquadrature cinematografiche. Tecnica che, come vedremo nei risultati comparando diverse prestazioni, riesce a migliorare le performance rispetto ad altri approcci con lo stesso obiettivo presenti in letteratura (e.g. utilizzo di una semplice CNN, una MLP o anche la classificazione solo tramite VGG16 o approcci senza quarto classificatore). L'approccio di Stacking Ensemble Learning prevede più step, il primo è stato già descritto e consisteva nell'addestrare tre VGG16 con la stessa architettura ma alimentati da dataset sostanzialmente diversi e conservare la loro architettura e i pesi ottimizzati al fine di poterli sfruttare per la tecnica di Ensemble (come descritto nell'excursus teorico sull'argomento, lo stacking learning è un modello eterogeneo, cioè composto da weak classifier diversi, ma nell'approccio presentato i classificatori deboli saranno gli stessi, solo il tipo di allenamento sarà differente). Il secondo passo, descritto successivamente, è la creazione di tre training set di nuove immagini (sempre suddivisi in immagini RGB, immagini segmentate e ipercolonne) i quali diverranno la base per l'addestramento di un quarto classificatore, vero strumento di classificazione di questo esperimento, posto dopo le tre VGG16. Il terzo e ultimo passo è proprio il training del suddetto meta-classificatore che in realtà non è addestrato direttamente sui training set ma sulle predizioni che le tre reti VGG16 fanno su di essi (array di predizioni).

5.3.3.1 Algoritmo di creazione dei training set per il quarto classificatore

Come innanzi evidenziato, al fine di procedere con la sperimentazione e l'implementazione della metodologia di Deep Learning scelta, lo stacking ensemble learning, bisogna costruire ex-novo tre training set contenenti nuove immagini per tutte le categorie di scatto cinematografico e tutte le tipologie di dataset viste (RGB, Image Segmentation, Hypercolumns). Questi tre nuovi training set fungeranno da campioni di addestramento per un quarto classificatore. In questa sperimentazione è stata scelta una semplice rete neurale Multilayer Perceptron. In realtà, come preannunciato, il training di questa rete finale non sarà effettuato in modo

tradizionale fornendo direttamente le immagini ma in input le verrà passato un array di predizioni performate dalle tre VGG16 sui tre nuovi training set di riferimento. Più specificamente, le reti convoluzionali pre-addestrate saranno impiegate per svolgere delle predizioni su tre immagini alla volta, una per ogni VGG, appartenenti ai tre set di dati. Le predizioni (che per la precisione sono probabilità di classificazione nelle diverse classi, dove la categoria con la percentuale più alta viene scelta come corretta) saranno poi combinate dal meta-classificatore al fine di ottenere un'accuratezza superiore sfruttando i punti di forza di ogni vgg16 e le prediction più accurate per ogni immagine. Per far sì che questo processo funzioni, i tre classificatori dovranno processare contemporaneamente la stessa immagine ma nelle tre versioni: RGB, segmentata e ipercolonnare. Questo aggiunge la necessità di avere lo stesso ordine per tutti e tre i training set creati ex-novo. Proprio per questo bisogno lo script di creazione dei set di addestramento, molto simile a quello usato precedentemente per lo stesso scopo, ha delle variazioni evidenti che garantiscono lo stesso ordine per tutti i set di dati creati. Vediamo il codice:

```
import numpy as np
import tensorflow.keras
import os
import cv2
from tensorflow.keras.utils import to_categorical

WIDTH = 160
HIGHT = 90

DatasetRGB_Stk= "C:/Users/Francesco/Desktop/DatasetRGB_Stacking/"
DatasetStz_Stk= ""C:/Users/Francesco/Desktop/DatasetStlz_Stacking"
DatasetHype_Stk= "C:/Users/Francesco/Desktop/DatasetHype_Stacking/"
```

Come prima cosa, oltre richiamare le librerie utilizzate e impostare le dimensioni con cui scaleremo le immagini, salviamo all'interno di variabili globali i path dei tre nuovi dataset creati ex-novo, i quali naturalmente hanno subito già l'intero processo descritto nel trattamento dei dati. Questa è la prima variazione rispetto

all'algoritmo descritto precedentemente che lavorava con un set di dati alla volta. Qui però, per andare incontro alla necessità di conservare un ordine unico per tutte le immagini, il lavoro di creazione dei training set è svolto contemporaneamente per tutti e tre i nuovi dataset.

```
CATEGORIE= ["CL", "CM", "FI", "MB", "MF","PA","PP","PPP"]

training_list = []

def create_training_data():
    for category in CATEGORIE:
        path1= os.path.join(DatasetRGB_Stk, category)
        path2= os.path.join(DatasetStz_Stk, category)
        path3= os.path.join(DatasetHype_Stk, category)
        class_num= CATEGORIE.index(category)

        for norm, stlz, hype in
zip(sorted(os.listdir(path1)),sorted(os.listdir(path2)),sorted(os.listdir(path3))):
            try:
                img_array1 = cv2.imread(os.path.join(path1,norm))
                new_array1 = cv2.resize(img_array1,(WIDTH, HIGHT))

                img_array2 = cv2.imread(os.path.join(path2,stlz))
                new_array2 = cv2.resize(img_array2,(WIDTH, HIGHT))

                img_array3 = cv2.imread(os.path.join(path3,hype))
                new_array3 = cv2.resize(img_array3,(WIDTH, HIGHT))



                training_list.append([new_array1,new_array2,new_array3,class_num])

            except Exception as e:
                pass
```

create_training_data()

Il cuore dello script è tutto in questa funzione Python appena definita. In sostanza questa resta molto simile a quella adoperata per creare i primi training set ma, essendo svolta la costruzione per tre dataset contemporaneamente al fine di soddisfare il "*must have"* di ordine coerente, il ciclo for varia di qualche linea di codice. In primis il ciclo è svolto su tutte e tre i path tramite il metodo *zip* che permette di ciclare su più liste senza fare dei loop annidati. Inoltre, grazie alla funzione *sorted* leggiamo ogni lista, derivata dalla concatenazione di path e categoria, in modo ordinato (sorted effettua l'ordinamento di una lista, ed essendo le tre liste formate da immagini uguali, l'ordine sarà lo stesso). Per il resto la funzione è esattamente identica a quella spiegata nei paragrafi precedenti con l'eccezione che ogni operazione è svolta tre volte e che la lista di training avrà nelle prime tre posizioni gli array contenenti le immagini dei tre diversi dataset e nell'ultima cella le etichette che sono uniche per tutte e tre i vettori e coerenti con l'ordine stabilito.

```
import random

random.shuffle(training_data)

training_imgs_norm=[]
training_imgs_stlz=[]
training_imgs_hype=[]
training_labels=[]

for norm,stlz,hype, label in training_data:
    training_imgs_norm.append(norm)
    training_imgs_stlz.append(stlz)
    training_imgs_hype.append(hype)
    training_labels.append(label)
```

```
training_labels=np.array(training_labels)
y=to_categorical(training_labels)
training_imgs_norm=np.array(training_imgs_norm).reshape(-1, 90, 160,3)
training_imgs_hype=np.array(training_imgs_hype).reshape(-1, 90, 160,3)
training_imgs_stlz=np.array(training_imgs_stlz).reshape(-1, 90, 160,3)

plt.imshow(training_imgs_norm[15])
plt.show()
plt.imshow(training_imgs_stlz[15])
plt.show()
plt.imshow(training_imgs_hype[15])
plt.show()
```

Anche questo estratto risulta essere di fondamentale importanza ai fini di un buon training del quarto classificatore. Nelle prime righe è svolta la solita randomizzazione dei campioni, operazione effettuata sull'intera lista contenente tutti e tre gli array di training e i label unici proprio per conservare l'ordine sia tra le immagini dei vari set di dati sia, naturalmente, con le relative etichette.
A valle di questo mescolamento dei dati la lista generale viene divisa in quattro nuove liste dove le prime tre conterranno i diversi set di training images e l'ultima rappresenterà invece le etichette univoche per tutti. Successivamente sono performate le solite attività di conversione in Numpy array, unico formato accettato dalla rete in fase di addestramento, di scaling e di cambio di codifica delle etichette in One Hot Encoding. Inoltre, al fine di verificare se l'ordine tra i tre set di immagini si è conservato, è stata "plottata", tramite la libreria matplotlib.pyplot, la stessa posizione dei tre array. Il risultato è visibile nella Fig. 5.14.

Figura 5.14: Plotting della quindicesima posizione dei tre array di immagini al fine di provare che l'ordine vettoriale è il medesimo

```
import pickle

pickle_out=open(""C:/Users/Francesco/Desktop/Img_N_Dataset2.pickle","wb")
pickle.dump(training_imgs_norm, pickle_out)
pickle_out.close()

pickle_out=open(""C:/Users/Francesco/Desktop/Img_S_Dataset2.pickle","wb")
pickle.dump(training_imgs_hype, pickle_out)
pickle_out.close()

pickle_out=open(""C:/Users/Francesco/Desktop/Img_H_Dataset2.pickle","wb")
pickle.dump(training_imgs_stlz, pickle_out)
pickle_out.close()

pickle_out=open(""C:/Users/Francesco/Desktop/All_Lab_StkLearning2.pickle","
wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

Naturalmente, per concludere lo script, i diversi output generati sono salvati in file pickle al fine di non dover riprodurre il processo. I file saranno successivamente riutilizzati nello script di Ensemble Learning in fase di addestramento del quarto classificatore. Tale procedimento sarà meglio, illustrato nell'ultimo paragrafo del capitolo.

5.3.3.2 Implementazione del modello di Stacking Ensemble Learning per la classificazione delle inquadrature cinematografiche

Prima di un bilancio finale dettato dai risultati e di un momento di *Data e Model Explanation* inserito per fornire al lettore una spiegazione più generale della complessità del problema affrontato, occorre illustrare brevemente l'ultimo step della metodologia di Deep Learning, lo stacking ensemble learning.
Come già ampiamente descritto a livello teorico, l'ensemble learning è una tecnica di "*boosting"* dei risultati di un problema di apprendimento automatico.
Questa metodologia, che combina più classificatori in un solo modello traendo tutti i vantaggi da ogni modello attraverso l'aggregazione delle predizioni, tende a bilanciare la varianza e la distorsione dei risultati derivanti dai dati e dagli algoritmi adoperati durante la prima classificazione, migliorando così l'accuratezza. In particolare, lo stacking learning preso qui in considerazione, è una versione particolare di ensemble perché è l'unico che si avvale di un ulteriore classificatore oltre i weak classifier. Questo quarto classificatore che. come già precisato, per questa sperimentazione sarà una rete neurale MLP, è posto a valle dei primi modelli e sarà lo strumento che combinerà i weak classifier, le nostre VGG16 pre-trainate su tre dataset differenti tramite transfer learning, aggregando gli output di classificazione dei dataset. Tale operazione è svolta in due step: il primo consiste nel fornire alle tre VGG16 un test set ciascuno di immagini mai processate (creati con lo script precedente e chiamati in quel frangente come training set perché in realtà sono la base del set di addestramento del quarto classificatore ma per i weak classifier risultano essere dei test set) su cui queste effettueranno le predizioni di classificazione e genereranno un array con tutte le prediction per tutte le immagini (array di probabilità). Al termine di tale processo, la cui tempistica varia in base al

volume dei test set forniti, si avranno tre vettori di predizioni che saranno poi riunificati in un solo grande array di prediction e diverranno il training set del quarto classificatore che le combinerà per avere il risultato più accurato (come accennato questa aggregazione funziona solo se la predizione per ogni immagine nelle tre differenti versioni si trova alla stessa posizione all'interno dei tre array ovvero se le tre VGG16 hanno processato contemporaneamente le stesse immagini nelle differenti versioni). Dopo questa breve ricapitolazione sull'architettura del modello e sul suo funzionamento, è possibile ora passare al lato più pratico e presentare il codice[118] che svolge queste operazioni arricchendolo di qualche commento esplicativo.

```
json_file = open('Norm_X_StkLrn.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
N_loaded_model = model_from_json(loaded_model_json)
json_file = open('Stlz_X_StkLrn.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
S_loaded_model = model_from_json(loaded_model_json)
json_file = open('Hype_X_StkLrn.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
H_loaded_model = model_from_json(loaded_model_json)

N_loaded_model.load_weights("Norm_X_StkLrn_weights.h5")
S_loaded_model.load_weights("Stlz_X_StkLrn_weights.h5")
H_loaded_model.load_weights("Hype_X_StkLrn_weights.h5")
print("Loaded norm model from disk")

N_loaded_model.compile(loss="categorical_crossentropy", optimizer="sgd",
metrics=["accuracy"])
S_loaded_model.compile(loss="categorical_crossentropy", optimizer="sgd",
metrics=["accuracy"])
```

[118] https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/

H_loaded_model.compile(loss="categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])

I primi passi di questo algoritmo, oltre all'import delle librerie, consistono nel ricaricare su delle variabili globali dello script implementato i tre file JSON su cui era salvata l'architettura delle VGG16 a valle del transfer learning. Naturalmente l'architettura delle reti non è sufficiente a far sì che le VGG siano pronte per essere adoperate come weak classifier dello stacking learning e, per tal motivo, i pesi di ogni rete salvati dopo l'addestramento (pesi generati dall'ottimizzazione che avviene durante la fase di training) sono ricaricati sui modelli contenenti le architetture *"plain"*[119] appena importate. Successivamente i tre modelli vengono ricompilati dal calcolatore riassegnando la funzione di costo, sempre la *categorical_crossentropy*, l'ottimizzatore che per semplicità non viene variato (sgd corrisponde allo Stochastic Gradient Descent) e la metrica di performance scelta, anch'essa sempre la stessa in modo da poter confrontare tutti i risultati su una stessa scala. Arrivati a questo punto abbiamo tre classificatori già addestrati e pronti per svolgere il loro lavoro di predizione. Quello che ancora manca per svolgere questa prima fase di stacking learning sono i nuovi set di immagini ordinati e le relative etichette corrette. operazione effettuata proprio dalle seguenti righe di codice:

Xn=pickle.load(open("Img_N_Dataset2.pickle","rb"))
yn=pickle.load(open("All_Lab_StkLearning2.pickle","rb"))

Xn=Xn/255.0

Xs=pickle.load(open("Img_S_Dataset2.pickle","rb"))
ys=pickle.load(open("All_Lab_StkLearning2.pickle","rb"))

Xs=Xs/255.0

Xh=pickle.load(open("Img_H_Dataset2.pickle","rb"))

[119] Per architettura *plain* si intende lo scheletro del modello senza pesi calcolati

```
yh=pickle.load(open("All_Lab_StkLearning2.pickle","rb"))

Xh=Xh/255.0
```

Come già sottolineato, la modalità di ricarica di un set di immagini già creato e dei corrispondenti label avviene sempre tramite file pickle, aperti, letti e salvati dentro una variabile dello script. Questo avviene per tutti e tre i nuovi set di dati sia per le immagini che per le relative etichette. A valle del caricamento, le immagini passano sempre per un processo di normalizzazione al fine di ridurre il carico computazionale senza perdita di informazioni (X=X/255.0).

```
in_n=N_loaded_model.predict(Xn)
in_s=S_loaded_model.predict(Xs)
in_h=H_loaded_model.predict(Xh)

in_n=np.array(in_n)
in_s=np.array(in_s)
in_h=np.array(in_h)

in_tot=[]

for a in range(len(Xn)):
    in_tot.append([in_n[a], in_s[a], in_h[a]])

in_tot=np.array(in_tot).reshape(-1,8,3)
```

L'estratto di script appena inserito ha una valenza enorme per tutto il modello perché è ciò che differenzia questo approccio dall'addestramento tradizionale di un classificatore o dalle altre tipologie di Ensemble Learning (bagging, boosting...) poiché qui come training set non saranno fornite delle immagini ma una set di predizioni per ogni dato. Nelle prime righe dello script le VGG16 pre-trainate effettuano il compito per cui sono state create e addestrate, ossia predire in modo

automatico la classificazione di un determinato set di immagini. Naturalmente ogni vgg effettuerà tale lavoro sul set di dati coerente con le immagini su cui è stata allenata, pertanto la prima lo svolge sui dati RGB, la seconda sui dati segmentati e la terza sulle Ipercolonne. Questa operazione ha un tempo variabile che dipende dal volume dei campioni. Dopo la creazione degli array di predizioni da parte delle reti, questi sono convertiti in Numpy per garantire la compatibilità con la funzione *fit* dei modelli in Keras. A seguire bisogna creare la collezione completa di dati da fornire al quarto classificatore come training set; per questo motivo i prediction array vengono riunificati in unica grande lista tramite un ciclo for della lunghezza di uno degli array di training (contenendo ogni array lo stesso numero di predizioni, per la precisione 1266, poiché i tre test set avevano lo stesso volume dati, la scelta del vettore su cui ciclare è ininfluente sul risultato). Ogni cella di tale lista corrisponderà alle tre predizioni della stessa immagine nelle diverse versioni (motivo per il quale serviva un ordinamento unico). Per concludere, dopo la creazione di quello che sarà il training set del classificatore, la lista di predizioni verrà trasformata in un Numpy array per essere fornita in addestramento alla rete MLP.

x_train, x_test,y_train, y_test=train_test_split(in_tot, yn, stratify = yn, test_size = 0.1)

Il codice mostrato in questo estratto ha una funzione essenziale per ottenere risultati accurati e validati. Esistono diversi modi per avere una convalida dell'accuratezza del training ma uno dei più efficaci, insieme alla accuracy sul validation set utilizzata per le VGG16, è quello di creare un parametro di confronto, effettuando delle predizioni su set di dati sconosciuto alla rete. Attraverso il metodo *train_test_split* è possibile costituire un campione di test suddividendolo percentualmente dal nostro training set di predizioni, il quale non verrà più utilizzato per la fase di allenamento ma solo per una valutazione finale dell'accuratezza. Ovviamente vale lo stesso per i labels correlati. In questa sperimentazione viene usato il 10% del set di addestramento che corrisponderà con la dimensione di un test set adatto a valutare le prestazioni finali della rete

```
model=Sequential()

model.add(Dense(64, activation='relu',input_shape=in_tot.shape[1:]))
model.add(Flatten())
model.add(Dense(32,activation='relu' ))
model.add(Dense(8, activation='softmax'))
model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=['accuracy'])
hist=model.fit(x_train,y_train, epochs=20)
model.evaluate(x_test, y_test)
```

È in queste ultime righe di codice che avviene la "magia" dello Stacking Ensemble Learning. Prima di tutto viene creata l'architettura del quarto classificatore, una semplice MLP di tipo feedforward con tutti layer fully connected: il Flatten che ridimensiona e appiattisce il set di predizioni multidimensionali trasformandole in un *one-dimension* array, il dense che funge da attivatore della classificazione anche attraverso la funzione ReLU la quale introduce la non linearità nel processo e infine il softmax che svolge effettivamente il ruolo di classificatore della rete dando una valore di probabilità delle diverse classi per ogni immagine e scegliendo quella più corretta in base ai dati provenienti dai layer precedenti. Quindi vengono svolte le solite tre operazioni di addestramento di una rete, il compile, il fit (questa volta solo con 20 epoch, abbastanza per svolgere il training di una rete con architettura semplice e dati in input meno complessi delle immagini) ed infine evaluate che ci fornisce il nostro risultato finale, la stima della bontà del modello ovvero l'accuratezza del modello su un set di dati mai processato. Il processo, così come illustrato, sembra non differire molto dalla fase di addestramento spiegata per le VGG16. In realtà ciò che succede in questa training phase è molto più complesso e statisticamente di valore perché il quarto classificatore combinerà le tre predizioni che gli vengono fornite per ogni immagine e farà in modo di ottenere il risultato migliore prendendo la predizione più veritiera e accurata. In tal modo tale metodologia migliora le performance di classificazione delle inquadrature cinematografiche rispetto ad una semplice VGG16 o ad altri approcci e riduce notevolmente la possibilità di Overfitting, problema che affligge molti altri modelli

testati in letteratura. Con questo ultimo algoritmo si conclude la fase di modellistica e di sviluppo, i risultati di questa nuova metodologia di Deep Learning applicata alla classificazione degli scatti cinematografici e il confronto con le prestazioni della VGG16 saranno presentati, spiegati e discussi nel successivo capitolo.

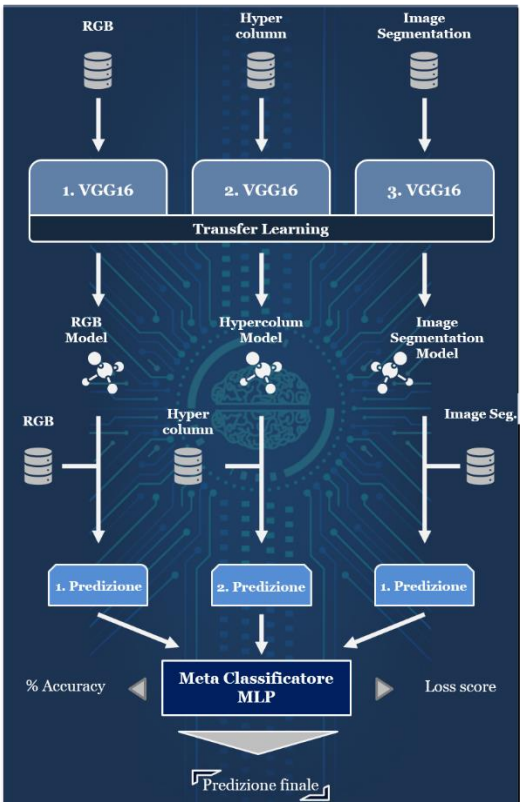


Figura 5.15: Rappresentazione dell'approccio di Stacking Ensemble Learning implementata

Capitolo 6: RISULTATI E VISUAL EXPLANATION

Dopo la costruzione del modello di Stacking Ensemble Learning e l'implementazione di tutti gli step per giungere all'obiettivo di classificare in modo accurato le inquadrature cinematografiche, è ora possibile mostrare le performance ottenute da questo nuovo approccio testato. Il capitolo 6 seguirà la stessa sequenza metodologica del precedente, pertanto non saranno presentati direttamente i risultati finali ma si procederà ad analizzare tutto il percorso compiuto partendo dalle performance delle tre fine tuned VGG16 in fase di addestramento (esaminandone una alla volta poiché le loro prestazioni sono differenti essendo allenate su set di versioni diverse delle stesse immagini) e, solo successivamente, saranno analizzate le prestazioni del modello finale di Ensemble Learning durante il processo di training del quarto classificatore tramite set di predizioni. Tali performance saranno analizzate (anche tramite aiuto visivo), commentate e messe a confronto con altre metodologie al fine di avvalorare i risultati della tesi. Le simulazioni di addestramento delle reti VGG16 e del modello di Stacking sono state implementate tramite il cluster Big Data Lab del Politecnico di Torino che ha permesso una processo più rapido ed efficiente e una fase di sperimentazione ottimizzata.

A conclusione del capitolo, prima delle considerazioni finali derivanti dai risultati qui presentati, è inserita una fase di *Visual Black Box Model Explanation* in cui verrà valutato, a livello visivo, tramite l'algoritmo LIME[17] di *Local Explainable AI* molto all'avanguardia per *l'image processing*, come i modelli appartenenti alla nostra metodologia lavorano ed effettuano le predizioni di classificazione (come già spiegato nella sezione teorica dedicata all reti neurali, queste tipologie di modelli sono tanto efficienti al livello di performance e *decision-making* quanto difficili da comprendere per quanto riguarda la modalità con cui i risultati sono raggiunti, questo perché ciò che accede dentro la sezione hidden delle reti è di difficile visualizzazione e, quindi, analisi). La tecnica LIME è stata utilizzata perché integrabile localmente a qualsiasi modello di apprendimento automatico che effettua predizioni e quindi utilizzabile per dei confronti di comportamento tra i diversi modelli su più tipologie di inquadrature. In breve, ciò che LIME persegue è

comprendere cosa succede all'interno della rete black box quando questa riceve dei dati in input e quali sono gli elementi che più influenzano le sue scelte, tutto ciò a livello visivo. Questo viene effettuato tramite la perturbazione dell'input fornito al modello e l'analisi dei delta di variazione delle predizioni performate.

6.1 **Risultati sperimentali di classificazione delle inquadrature**

In questo primo paragrafo dell'ultimo capitolo dell'elaborato, come già spiegato nell'introduzione, saranno presentati e analizzati i risultati ottenuti durante la fase di testing della metodologia scelta per l'ottimizzazione della classificazione delle inquadrature, volta alla creazione di un tool a supporto per gli addetti ai lavori in ambito creativo e cinematografico come il video editing, il trailer making, la photo set optimization e la movie smart reccommendation. Per concludere, i risultati saranno messi a confronto con altre tecniche presenti in letteratura, testate nello stesso ambito e per lo stesso scopo al fine di mostrare i miglioramenti ottenuti, individuare i gap e le carenze di tutti i modelli e provare a immaginare possibili next step da applicare alla metodologia in uso o anche approcci futuri potenzialmente più performanti.

6.1.1 *Risultati prima classificazione tramite VGG16*

Il percorso di esposizione dei risultati ottenuti tramite metodologia all'avanguardia di Deep Learning parte dai test di classificazione degli scatti cinematografici effettuati adoperando le reti convoluzionali VGG16 ri-trainate su tre dataset differenti (composto ognuno da 9279 immagini) appositamente costruiti (RGB, Image Segmentation, Hypercolumns), grazie alla tecnica di transfer learning. Le tre reti addestrate, come già ampiamente illustrato, sono parte dell'approccio innovativo scelto essendo protagoniste della creazione del training set di un quarto classificatore, strumento principale dello stacking ensemble learning.
Entrando più nel merito, come è possibile osservare dalla Fig.6.1, tutte le tre reti VGG16 addestrate sui tre training set derivanti da collezioni di dati diverse (ognuno composto da 8907 immagini equamente suddivise nelle otto categorie) e con valore

di epoch uguale a 40, hanno raggiunto un'accuratezza in fase di classificazione del set di allenamento vicina al 99%, una precisione che sembra eccezionale ma con delle riserve che vedremo a breve.

```
Epoch 35/40
8907/8907 [==============================] - 556s 62ms/sample - loss: 0.0474 - accuracy: 0.9926 - val_loss: 1.0779
- val_accuracy: 0.6855
Epoch 36/40
8907/8907 [==============================] - 556s 62ms/sample - loss: 0.0662 - accuracy: 0.9878 - val_loss: 1.1175
- val_accuracy: 0.7796
Epoch 37/40
8907/8907 [==============================] - 551s 62ms/sample - loss: 0.0616 - accuracy: 0.9887 - val_loss: 1.0267
- val_accuracy: 0.7957
Epoch 38/40
8907/8907 [==============================] - 561s 63ms/sample - loss: 0.0487 - accuracy: 0.9929 - val_loss: 1.0280
- val_accuracy: 0.7742
Epoch 39/40
8907/8907 [==============================] - 552s 62ms/sample - loss: 0.0376 - accuracy: 0.9937 - val_loss: 0.9252
- val_accuracy: 0.7930
Epoch 40/40
8907/8907 [==============================] - 553s 62ms/sample - loss: 0.0303 - accuracy: 0.9969 - val_loss: 0.9539
- val_accuracy: 0.7527

Epoch 35/40
8907/8907 [==============================] - 565s 63ms/sample - loss: 0.0412 - accuracy: 0.9948 - val_loss: 1.3496
- val_accuracy: 0.7043
Epoch 36/40
8907/8907 [==============================] - 560s 63ms/sample - loss: 0.0437 - accuracy: 0.9946 - val_loss: 1.0463
- val_accuracy: 0.7151
Epoch 37/40
8907/8907 [==============================] - 568s 64ms/sample - loss: 0.0352 - accuracy: 0.9964 - val_loss: 1.3887
- val_accuracy: 0.7043
Epoch 38/40
8907/8907 [==============================] - 563s 63ms/sample - loss: 0.0370 - accuracy: 0.9962 - val_loss: 1.1620
- val_accuracy: 0.7366
Epoch 39/40
8907/8907 [==============================] - 566s 64ms/sample - loss: 0.0184 - accuracy: 0.9993 - val_loss: 1.3347
- val_accuracy: 0.7231
Epoch 40/40
8907/8907 [==============================] - 564s 63ms/sample - loss: 0.0140 - accuracy: 0.9998 - val_loss: 1.4255
- val_accuracy: 0.7231

Epoch 35/40
8906/8906 [==============================] - 560s 63ms/sample - loss: 0.1252 - accuracy: 0.9782 - val_loss: 1.4989
- val_accuracy: 0.7285
Epoch 36/40
8906/8906 [==============================] - 554s 62ms/sample - loss: 0.1213 - accuracy: 0.9778 - val_loss: 1.4063
- val_accuracy: 0.7500
Epoch 37/40
8906/8906 [==============================] - 570s 64ms/sample - loss: 0.1022 - accuracy: 0.9830 - val_loss: 1.3287
- val_accuracy: 0.7285
Epoch 38/40
8906/8906 [==============================] - 545s 61ms/sample - loss: 0.0992 - accuracy: 0.9842 - val_loss: 1.4668
- val_accuracy: 0.7366
Epoch 39/40
8906/8906 [==============================] - 556s 62ms/sample - loss: 0.0822 - accuracy: 0.9879 - val_loss: 1.4180
- val_accuracy: 0.7204
Epoch 40/40
8906/8906 [==============================] - 552s 62ms/sample - loss: 0.1049 - accuracy: 0.9820 - val_loss: 1.4891
- val_accuracy: 0.7285
```

Figura 6.1: Performance delle reti VGG16 in fase di classificazione dei training set e dei validation set provenienti dai tre set di dati costruiti: 1) RGB 2) Image Segmentation 3) Hypercolumns

Nella Fig. 6.1 sono presenti, nella penultima e ultima riga per ogni epoch, anche le prestazioni delle tre reti in fase di classificazione del validation set (loss e accuracy). Questa operazione è effettuata per convalidare i risultati ottenuti durante il training e verificare la presenza di problemi in fase di addestramento come l'Overfitting o l'Underfitting. Ciò avviene confrontando i valori di accuratezza raggiunti dalle VGG16 durante la classificazione del training set (learning phase) con le prestazioni ottenute in predizione su immagini mai processate e non facenti parte dei dati di

training (il validation set dunque, che in questo caso è un campione di 372 immagini suddivise nelle diverse categorie di inquadrature). L'Overfitting e l'Underfitting sono delle problematiche in cui si può incorrere durante l'addestramento di una rete neurale, entrambe sono relative alla difficoltà da parte del modello di generalizzare l'apprendimento e quindi di riuscire ad applicare con la stessa precisione di risultato ciò che hanno imparato nella learning phase su campioni di immagini che non fanno parte del set di training, rendendole così inutilizzabili in campo pratico.
Tornando alla Fig. 6.1 si nota che le valutazioni di tutti i tre modelli (accuracy sul validation set) sono percentualmente abbastanza più carenti rispetto al 99% che tutte le reti raggiungono sul training set. Vediamo che la prima VGG16, addestrata sulle immagini RGB, supera in media il 75% (parliamo di media perché le prestazioni cambiano in base alla sequenza di campioni che la rete riceve in entrata e quindi, eseguendo più volte il processo, si ottengono risultati di addestramento leggermente diversi) e risulta essere la più performante delle tre, probabilmente perché allenata su un set di immagini normali, non manipolate da altri algoritmi e quindi complete di tutti gli elementi differenziatori delle diverse categorie, elementi che probabilmente sono stati trascurati o eliminati durante le varie trasformazioni al fine di esaltare feature diverse (e.g. durante la conversione delle immagini in dati segmentati, gli elementi rappresentanti la figura umana sono stati messi in risalto al fine di classificare categorie come la mezza figura, il primo piano ecc., questo però ha portato alla perdita di informazioni sul background, elemento differenziatore dei campi lunghi e i campi medi). La rete VGG allenata sul training set formato dalla collezione di immagini segmentate raggiunge mediamente il 70% in classificazione sul validation set e risulta essere la meno prestante delle tre per i motivi suddetti ma, nonostante ciò, gode di alcuni punti di forza sulla classificazione di alcune categorie molto utili al modello finale. La terza e ultima VGG16, la quale ha ricevuto in input durante la fase di training le immagini ipercolonnari, ottiene un'accuracy sul set di validazione del 72% (naturalmente sempre in media) e quindi si posiziona a metà tra le due precedenti. Questa performance migliore dell'altro set di dati trasformato è probabilmente dovuta alla proprietà di *Edge detection* di cui godono le Hypercolumns che permette di mettere sì in risalto i bordi delle figure umane ma anche gli elementi sullo sfondo. I risultati sono invece meno ottimali rispetto ad una rete addestrata sulle immagini RGB quasi certamente per la perdita

di saturazione delle immagini trasformate che riduce l'informazione sulle inquadrature acquisite dalla rete tramite i colori.

Come abbiamo osservato, tutte le reti utilizzate e addestrate con set di dati diversi hanno una prestazione ottima sul training set ma una netta differenza di performance quando si ritrovano a classificare nuove immagini; ciò significa che ci troviamo di fronte ad un sostanziale problema di Overfitting (l'Underfitting solitamente implica che ci sia sì una differenza di risultati tra classificazione del training set e del validation set ma anche che l'accuracy in fase di addestramento sia molto bassa e che quindi la rete non riesca proprio ad apprendere, probabilmente a causa di un modello troppo semplice per il tipo di dati e non è questo il caso). Esistono diverse metodologie per superare il problema di Overfitting e di mancanza di generalizzazione dell'apprendimento: un metodo relativamente semplice ma molto *time consuming* consiste nell'aumento del set di dati o tramite la classificazione di nuove immagini (approccio migliore al fine di fornire campioni variegati utili al training ma anche operazione molto dispendiosa) o attraverso l'utilizzo della tecnica di *Data Augmentation* (espansione del set di dati grazie alla generazione di diverse versioni delle stesse immagini ma con piccole variazioni come il cambio della tonalità, la rotazione ecc.) che però ha lo svantaggio di creare dati troppo simili tra loro e con poca informazione aggiuntiva. In questo elaborato invece, il problema è stato risolto utilizzando lo stacking ensemble learning che, oltre a migliorare i risultati a livello di performance (obiettivo principale), tende a stabilizzare l'apprendimento, ridurre la differenza di prestazioni tra classificazione su training e validation set e quindi ad eliminare l'Overfitting (come vedremo successivamente nella sezione dedicata ai risultati finali).

Dopo questa prima breve analisi dei risultati generali raggiunti con i primi tre classificatori, entriamo più nel dettaglio passando dall'accuratezza di classificazione complessiva dei modelli osservata fino ad ora all'accuracy per categoria di scatto cinematografico, al fine di approfondire meglio le performance di ogni VGG16 utilizzata e mostrare i punti di forza di ogni modello. Per svolgere tale compito è fornito al lettore uno strumento visivo che mostra la bontà di classificazione per ogni classe, la confusion matrix (mostrate in Fig. 6.2 per tutte e tre le reti). Questa vista è implementata tramite la libreria di machine learning *sklearn.metrics*.

a)

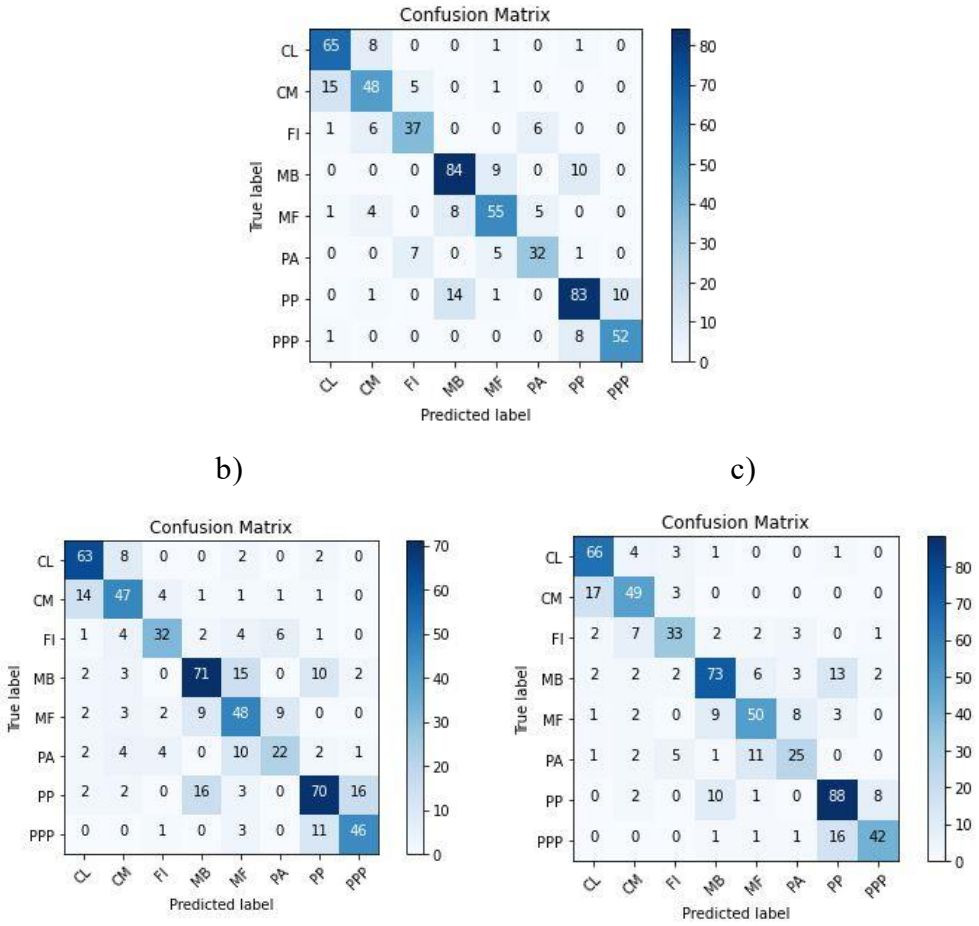


Figura 6.2: Confusion Matrix relative ai tre modelli VGG16:
a) RGB b) Image Segmentation c) Hypercolumns

La confusion matrix [120](anche chiamata error matrix) è una tecnica molto usata in ambito di machine learning e statistico per visualizzare le performance di un algoritmo di Supervised learning come quelli adoperati in questo elaborato. In breve, ogni riga presente all'interno della matrice corrisponde alle classi corrette in cui si vuole classificare, invece ogni colonna prende in considerazione le stesse categorie ma questa volta predette dal modello. Tale sistema a due dimensioni permette di comprendere se l'algoritmo è in confusione su alcuni campioni tra una classe e l'altra (da qui il nome confusion matrix). Osservando la Fig. 6.2 dove sono presentate le matrici derivanti dalla classificazione svolta da ogni VGG16 sul proprio validation set (in alto il modello allenato sui dati RGB, sulla sinistra image segmentation e a destra Ipercolonne) vediamo che sulle diagonali è mostrato il numero di immagini per categoria correttamente classificato (true label corrispondente al predicted label del modello). Come già intuibile i risultati

[120] https://en.wikipedia.org/wiki/Confusion_matrix

complessivi sono concordi con l'accuracy già osservata e mostrano che la VGG16 addestrata sul campione RGB performa meglio delle altre per i motivi già descritti, lo stesso per le restanti. Se invece si prova a guardare i picchi è possibile trovare i punti di forza e di debolezza di ogni rete ma anche comprendere quali sono le classi più difficili da classificare a causa della loro somiglianza visiva. Andando nel dettaglio, si nota che le classi meglio allocate in modo assoluto da tutte e tre le reti sono il primo piano, il mezzo busto e il campo lungo. I picchi ci dicono quali sono le categorie di inquadrature meglio classificate in modo assoluto tuttavia, non essendo tutte le classi rappresentate con la stessa cardinalità all'interno del validation set (anche se la numerosità delle classi all'interno dei dataset è equilibrata alcune categorie hanno più campioni e questa differenza è ampliata quando viene estratto il validation set in percentuale dal training set), per valutare in modo razionale quali sono le inquadrature su cui la rete è più esperta bisogna normalizzare e calcolare la percentuale di corretta classificazione per categoria. In questo modo troviamo che per la rete RGB effettivamente il campo lungo (87%) e il mezzo busto (81%) sono tra le classi in cui la VGG16 performa meglio ma una differenza rispetto ai valori assoluti la troviamo per il primo piano che percentualmente risulta avere solo un valore del 76% di successo che viene abbondantemente superato dalla percentuale di corretta classificazione del primissimo piano, 85%. Tale risultato è di facile interpretazione ma vale solo per questa rete, la principale delle tre, che lavora con immagini non trasformate ma solo pre-processate. In pratica suddividendo le otto classi di inquadrature in tre macro tipologie, la prima focalizzata su background con la figura umano di poco rilievo (campo lungo e campo medio), la seconda in cui la sagoma umana è messa in risalto ma tagliata (mezzo busto, mezza figura e piano americano) e la terza dove in evidenza c'è il volto umano (primo piano e primissimo piano), possiamo dire che il campo lungo, il mezzo busto e il primissimo piano sono le più caratteristiche, riconoscibili, delineabili e soprattutto le più estreme rispetto alle restanti della stessa macro tipologia. Questa motivazione le rende le classi con le quali è più facile non fare "confusione" in fase di classificazione. Discorso diverso per le altre due reti che sono utilizzate più per essere a supporto del modello alimentato con immagini RGB individuando determinati pattern di dati specifici che potrebbero essere poco individuabili dalla prima rete a causa della grossa variabilità dei campioni a colori portando ad una classificazione errata (come vedremo i due modelli aggiuntivi

insieme al quarto classificatore di stacking learning serviranno a fare boost delle performance tramite predizioni più accurate solo su alcuni campioni che potrebbero essere mal interpretati invece dalla rete principale). Questo giustifica quindi da una parte un'accuracy minore sia in generale, sia per categoria di inquadratura rispetto alla VGG addestrata sui dati saturati e, dall'altra, avere delle categorie diverse nella *top three* di quelle con più successo in classificazione. Ad esempio, per la rete di Ipercolonne il campo lungo (88%) e il primo piano (81%) sono confermate come classi più performanti per il modello non solo in valore assoluto, è invece il campo medio ad assicurarsi il terzo posto con il 74%. Questo può derivare dalle caratteristiche delle Ipercolonne che mettono in risalto le texture dello sfondo e dei volti umani. Passando alla rete di image segmentation, questa ha ottimi risultati sui campi lunghi (84%) perché, mancando la figura umana, che le immagini in forma segmentata tendono a mettere in risalto, il dato relativo al campo lungo sarà solo un'immagine vuota in colore viola. Le altre due categorie con più successo sono il primissimo piano e il mezzo busto proprio per la capacità di esaltazione delle figure umane che caratterizza tali immagini. Riassumendo, in valori normalizzati le tre categorie con una migliore percentuale di classificazione in media tra le tre reti sono il campo lungo, il primissimo piano e il mezzo busto. La meno performante è invece il piano americano per la sua intrinseca somiglianza sia con la figura intera che con la mezza figura. Un'ultima considerazione riguarda le coppie di classi in cui avviene maggiormente lo scambio errato di dati. Data la somiglianza notiamo che spesso il campo lungo e il campo medio sono confusi dalle reti a causa della loro minima differenziazione, determinata solo dalla distanza della cinepresa. La medesima cosa accade tra primo piano e primissimo piano poiché in quest'ultimo il volto è solo più ravvicinato. Per quanto riguarda le classi centrali, possiamo osservare due gruppi di confusione: il primo riguarda il mezzo busto e la mezza figura e il secondo la figura intera e la mezza figura, molto simili tra loro a coppie e diversi solo per il punto in cui l'inquadratura taglia la figura umana e quindi molto fraintendibili dall'algoritmo. Per questo motivo di intrinseca vicinanza visiva, molti lavori svolti in questo ambito prendono in considerazione solo due, quattro o sei classi di inquadrature accorpando le restanti. In questo elaborato si è preferito estendere comunque l'analisi ad otto sia da un punto di vista di applicabilità al reale mondo del cinema, trattandosi delle inquadrature standard, sia per rendere il traguardo di una classificazione efficiente e accurata, competitivo e innovativo.

6.1.2 *Risultati finali classificazione tramite Stacking Ensemble Learning*

A valle dei risultati di classificazione delle inquadrature cinematografiche raggiunti dalle tre diverse VGG16 (RGB, Image Segmentation, Hypercolumns), è possibile ora presentare le performance finali della metodologia di Deep Learning (lo Stacking Ensemble Learning) implementata in questo elaborato per adempiere al compito di migliorare l'accuracy rispetto ad altri approcci presenti il letteratura (a tal fine sarà fornito anche un confronto con altri modelli e altri algoritmi di classificazione). Il punto di partenza per questa esposizione finale è la prestazione delle VGG16 ri-addestrate tramite transfer learning. Come già evidenziato, i tre modelli si pongono in una regione di accuracy sul validation set tra il 70% e il 75%, ampiamente discostato dalla loro performance sul training set che si aggira intorno al 99%. Questi risultanti denotano un importante gap di prestazioni e un evidente ma abbastanza normale Overfitting sui dati. L'utilizzo dello stacking learning ha permesso di eliminare questo effetto di difficoltà di generalizzazione dell'apprendimento e ci ha fornito un nuovo traguardo per quanto riguarda l'accuratezza di classificazione del set di dati mai processato (l'unica effettiva metrica comparativa ottimale). Andando più in dettaglio nella presentazione dei risultati, le nuove performance di classificazione sono state ottenute su un campione di 1266 immagini per ogni dataset, equamente suddiviso nelle varie inquadrature. I risultati sono stati generati dall'addestramento di 20 epoch di una rete Multilayer Perceptron alimentata dalle predizioni generate dalle VGG16 sui tre training set da più di 1200 campioni.

```
Epoch 15/20
1139/1139 [==============================] - 0s 193us/sample - loss: 0.3094 - accuracy: 0.8771
Epoch 16/20
1139/1139 [==============================] - 0s 308us/sample - loss: 0.3040 - accuracy: 0.8885
Epoch 17/20
1139/1139 [==============================] - 0s 177us/sample - loss: 0.2977 - accuracy: 0.8885
Epoch 18/20
1139/1139 [==============================] - 0s 192us/sample - loss: 0.2980 - accuracy: 0.8867
Epoch 19/20
1139/1139 [==============================] - 0s 189us/sample - loss: 0.2950 - accuracy: 0.8841
Epoch 20/20
1139/1139 [==============================] - 0s 188us/sample - loss: 0.2912 - accuracy: 0.8982
127/127 [==============================] - 0s 2ms/sample - loss: 0.3232 - accuracy: 0.9055

[0.3231920846334593, 0.9055118]
```

Figura 6.2: Prestazione simulazione di addestramento quarto classificatore MLP all'interno della tecnica di Stacking Learning

Ciò che risulta evidente dalla Fig. 6.2 è il netto miglioramento di prestazioni sul test set con un accuracy in fase di classificazione che supera il 90% in media (questo valore oscilla tra l'87% e il 92% in base alle simulazioni e alla sequenza di predizioni che il quarto classificatore riceve in input). Questo risultato è un gran traguardo che pone tale metodologia al di sopra delle altre presenti in letteratura che si posizionano sul confine del 75%. Alta evidenza osservabile dalla Fig. 6.2 è la completa eliminazione del problema di Overfitting, denotato invece durante l'addestramento delle VGG16, dimostrata dall'assenza di gap, se non per qualche punto percentuale colmato in altre simulazioni, tra accuracy sul training set e sul test set. Come altro vantaggio derivante dall'utilizzo di tale metodologia quindi, abbiamo l'aumento di stabilità dell'addestramento che si riflette in una migliore classificazione. Il lettore può notare che l'accuratezza in fase di training, nella stabilizzazione, è scesa dal 99% prodotto dalle VGG16 ad un valore poco superiore all'90%. Questa flessione a livello prestazionale è probabilmente dovuta al volume ridotto di campioni forniti sia alle VGG16 (9000 immagini non è ancora una cardinalità tale da poter raggiungere prestazioni eccellenti) sia alla rete MLP (1200 campioni) e quindi migliorabile in modo semplice ripetendo la fase di costituzione dei dataset e di etichettatura delle immagini (attività molto time consuming e che quindi si rimanda a possibili sviluppi futuri). Inoltre, questo calo di accuracy sul set di addestramento non è un grande problema perché è comunque preferibile un modello stabile, in cui le performance sul training set e sul test set coincidano, più che una metodologia affetta da Overfitting e quindi inutilizzabile a livello pratico (una rete che performa male su immagini mai processate è un modello poco utile anche se raggiunge alte percentuali set di addestramento). Un altro step efficace per mostrare al lettore le performance dello Stacking Ensemble Learning in classificazione per ogni classe è la confusion matrix.

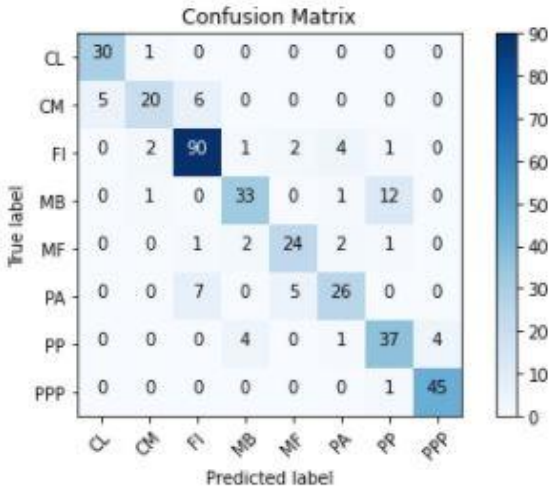


Figura 6.3: Confusion matrix delle performance di classificazione dello Stacking Ensemble Learning

A primo impatto notiamo i netti miglioramenti di classificazione, a livello percentuale, su ogni classe di inquadratura con dei picchi sul primissimo piano (98%) e campo lungo (97%) essendo le due categorie più estreme. Un netto miglioramento è percepibile anche sulla mezza figura (85%) invece il piano americano, data la sua struttura molto a cavallo tra due classi, resta la più difficile da classificare (72%, risultato comunque migliorato).

Naturalmente questo è solo un primo passo verso la giusta direzione e mediante alcuni correttivi migliorativi della metodologia (come l'aumento di cardinalità del volume dei dati) e approcci similari (e.g. Federated Learning) è sicuramente possibile potenziarla ulteriormente per renderla applicabile al campo cinematografico, in sinergia con altre classificazioni automatizzate (colori, colonne sonore, attori, registi), al fine di creare uno strumento di supporto agli addetti ai lavori (video editing, photoset Optimizing, streaming movie smart reccommendation, trailer making).

6.1.3 *Confronto con altri approcci presenti in letteratura*

Il modo più semplice per convalidare i risultati raggiunti, valorizzare il traguardo e valutare la robustezza della metodologia implementata è effettuare un confronto di

performance tra approcci e modelli diversi utilizzabili o presenti in letteratura per soddisfare l'obiettivo di una classificazione accurata delle inquadrature cinematografiche. Attraverso tale confronto si cercherà di dimostrare che l'approccio messo in atto, in due step di classificazione con un quarto classificatore a valle di tre reti neurali convoluzionali profonde, sia la scelta più adatta rispetto all'utilizzo di modelli lineari o approcci senza quarto classificatore.

Una prima comparazione è naturalmente con alcune tecniche presenti in letteratura meno complesse e raffinate di quella messa in atto in questo elaborato: una semplice Multilayer Perceptron e una semplice CNN. La rete MLP, costituita da soli livelli fully connected (un flatten e tre dense layer: 64, 32 e 8 nodi), addestrata sulle 7306 immagini RGB su ben 70 epoch (il training set è minore perché tale prova è stata effettuata prima dell'aumento dei campioni, invece il numero di epoch è stato aumentato di molto proprio per denotare quanto questa tipologia di rete sia di caratura inferiore rispetto ad altre), performa in modo deludente in classificazione del training set (80% di accuracy in media su 70 epoch) e in modo disastroso sul test set (maggiore del 30% in media) senza contare l'enorme Overfitting ottenuto. Questo è osservabile in Fig. 6.4.

```
Epoch 66/70
6721/6721 [==============================] - 9s 1ms/sample - loss: 0.5125 - accuracy: 0.8238
Epoch 67/70
6721/6721 [==============================] - 9s 1ms/sample - loss: 0.5127 - accuracy: 0.8221
Epoch 68/70
6721/6721 [==============================] - 9s 1ms/sample - loss: 0.4813 - accuracy: 0.8356
Epoch 69/70
6721/6721 [==============================] - 9s 1ms/sample - loss: 0.4721 - accuracy: 0.8402
Epoch 70/70
6721/6721 [==============================] - 9s 1ms/sample - loss: 0.4712 - accuracy: 0.8406
585/585 [==============================] - 0s 420us/sample - loss: 3.6258 - accuracy: 0.3402
Model Evaluation [3.6257784146528977, 0.34017095]
```

Figura 6.4: Performance di una simulazione di classificazione con una rete Multilayer Perceptron

La motivazione di tale scarsa prestazione in classificazione è insita nella tipologia di rete che è inadatta per svolgere la classificazione di dati high dimensional perché durante l'addestramento si ritrova a gestire un carico computazionale enorme dovuto alla creazione di valori sinaptici (pesi e bias) pari al numero di features e di collegamenti tra neuroni.

Questo problema è superato dalle reti CNN che sono più prestanti per l'image processing grazie al processo di convoluzione, pooling (che riduce la dimensionalità) e alla condivisione dei parametri tra alcuni nodi non fully connected

(le CNN hanno un numero di valori da gestire nettamente inferiore rispetto ad una MLP), per tal motivo è stata eseguita una simulazione anche con questo tipo di rete. La semplice CNN testata è formata da due livelli convoluzionali da 64 e 32 nodi entrambi con kernel 3x3, ognuno di questi è seguito da un layer di max pooling con passo 2x2 e infine è inserito un blocco di fully connected composto da un flatten e tre dense layer (64, 32 e 8 nodi). La rete è stata addestrata sempre con il medesimo set di dati della MLP ma su 50 epoch (sempre aumentate rispetto al nostro modello per testare se con un incremento di epoch il distacco di prestazioni rispetto al modello presentato potesse essere colmato).

```
Epoch 45/50
6721/6721 [==============================] - 187s 28ms/sample - loss: 0.5502 - accuracy: 0.9821
Epoch 46/50
6721/6721 [==============================] - 191s 28ms/sample - loss: 0.5873 - accuracy: 0.9688
Epoch 47/50
6721/6721 [==============================] - 193s 29ms/sample - loss: 0.6568 - accuracy: 0.9580
Epoch 48/50
6721/6721 [==============================] - 186s 28ms/sample - loss: 0.6888 - accuracy: 0.9613
Epoch 49/50
6721/6721 [==============================] - 188s 28ms/sample - loss: 0.6101 - accuracy: 0.9810
Epoch 50/50
6721/6721 [==============================] - 185s 28ms/sample - loss: 0.5926 - accuracy: 0.9747
585/585 [==============================] - 4s 7ms/sample - loss: 3.1317 - accuracy: 0.3744
Model Evaluation [3.1316792846744894, 0.37435898]
```

Figura 6.5: Performance di una simulazione di classificazione con una Convolutional Neural Network

Come da fig. 6.5, osserviamo un miglioramento dell'accuracy di classificazione del training set che riesce a raggiungere il 96% in media ma questo avanzamento non trascina le performance sul test set; queste ultime infatti si attestano sempre su una media del 35%. Qui il gap tra le due accuratezze è peggiorato creando più che un Overfitting un Underfitting probabilmente dovuto alla scarsa complessità della rete rispetto a quella dei dati.

Altra comparazione effettuata in modo rapido perché già illustrata e presentata è il test con una VGG16. La VGG16 è una rete molto complessa e raffinata e quindi risolve abbondantemente i problemi riscontrati con i confronti precedenti. Nell'ambito dell'analisi svolta in questa tesi la VGG16 addestrata sul dataset RGB su 40 epoch in modalità transfer learning non riesce a superare la soglia del 75% di accuracy sul validation set e genera un notevole Overfitting dovuto probabilmente al numero esiguo di dati. Questo basta per affermare che l'approccio utilizzato è superiore per la task assegnata rispetto a questa rete molto famosa.

Dopo aver confrontato e valutato le differenze prestazionali tra modelli lineari e il nostro approccio in due step, è stata effettuata una comparazione del nostro modello di Stacking Ensemble Learning con altri approcci di Ensemble learning, come il Bagging, che fanno a meno di un quarto classificatore ma combinano le predizione con metodi matematici. Il primo tra questi pone a valle delle tre VGG16, identiche a quelle dello stacking, un'aggregazione delle predizioni tramite "*soft voting*[121]". Questa tecnica, che prende in considerazione la media delle probabilità generate dai modelli, performa leggermente peggio rispetto alla metodologia di stacking raggiungendo un 65% in media. L'altra tecnica senza quarto classificatore testata si avvale dell'*hard voting* posto dopo le tre VGG16 alle quali viene dato lo stesso peso sulla decisione finale. In sostanza, i modelli voteranno l'esito della predizione e la classe che risulterà avere la maggioranza dei voti sarà considerata la decisione giusta. Le performance di tale tecnica sono le medesime di quelle del soft voting (tra il 60% e il 65%) e questo assunto ci induce a confermare che l'utilizzo di un quarto classificatore e, quindi dello Stacking Learning sia la scelta più performante e robusta sia rispetto a modelli lineari che rispetto ad altri approcci di Ensemble.

6.2 **Local visual explanation modello black box tramite LIME**

Al fine aumentare la comprensione e migliorare la spiegabilità dei modelli di Deep Learning adoperati e quindi comprendere e valutare i risultati di classificazione ottenuti e come questo algoritmo li ha raggiunti, è stata utilizzata LIME (Local Interpretable Model-agnostic Explanation) [17], una metodologia all'avanguardia di explanable AI. La tecnica LIME ci sarà d'aiuto nello spiegare come i dati forniti in input ai classificatori influenzano le predizioni ottenute. Questa metodologia sarà testata a valle delle tre VGG16 addestrate sui differenti dataset composti dalle versioni spiegate delle stesse immagini e ci permetterà di analizzare su che porzioni dell'immagine, i diversi modelli si basano per svolgere il compito di classificazione e apporre un label ad un determinato dato. In pratica quello che LIME [122]fornisce,

[121] Per soft voting si intende la tecnica di ricombinazione delle predizioni tramite la media delle probabilità generate dai modelli. La risposta scelta sarà quella con la migliore probabilità media

[122] https://towardsdatascience.com/interpretable-machine-learning-for-image-classification-with-lime

dopo aver ricevuto in input una rete addestrata e un campione di dati, è la regione dell'immagine che più è associata all'etichetta che la rete ha posto sul dato al termine della predizione. In output, come vedremo a breve, avremo quindi la stessa immagine data alla rete ma in cui è evidenziato, in diversi modi, il set di super-pixel [123]più caratteristici e che maggiormente hanno influenzato la classificazione. In tal modo, saranno quindi messe a confronto le tre reti VGG16 e analizzato cosa "vede" dei propri dati ognuno dei modelli implementati. In conclusione, questa procedura non solo aiuta a convalidare i risultati di predizione ma anche a comprendere se la metodologia creata risulta essere robusta per lo scopo per cui è utilizzata.

LIME permette di spiegare le predizioni a livello visivo sull'immagine generando nel suo algoritmo una variabile di *explanation* la quale crea un set di perturbazioni casuali, correlate con le relative predizioni, sul dato fornito in input. Queste perturbazioni del dato, effettuate attivando e disattivando alcune regioni di super pixel dell'immagine, andranno ad alimentare un surrogato di modello locale pesato, solitamente molto meno complesso di quello preso in considerazione in partenza e che gode di una buona interpretabilità (e.g. algoritmo di regressione). Questo andrà ad interpretare come tali perturbazioni modificano le previsioni del modello di partenza.

a)

b)

[123] I super pixel sono una regione di pixel caratterizzati dalle stesse feature. Questi sono molto adoperati nel campo della Computer Vision, Object detection e Image processing https://medium.com/@darshita1405/superpixels-and-slic

c)

d)

Figura 6.6: Campioni adoperati per l'analisi visuale performata da LIME appartenenti alle classi: a) Campo Lungo b) Mezzo Busto c) Primissimo Piano d) Piano Americano

Dopo questa breve spiegazione in merito al tool di local explanation, passiamo all'analisi pratica osservando i risultati ottenuti tramite LIME. Per questa analisi sono state prese in considerazione le immagini, una per ogni differente dataset, appartenenti alle tre classi (Campo Lungo, Mezzo Busto e Primissimo Piano) che in media le VGG16 classificano meglio con la finalità di osservare quali sono le regioni "*trigger*" delle diverse versioni dell'immagine che portano ad una corretta predizione. Sarà analizzata anche la classe su cui ogni modello performa con più difficoltà nell'inserimento nella giusta categoria, ovvero il Piano Americano. In Fig. 6.6 i campioni di partenza.

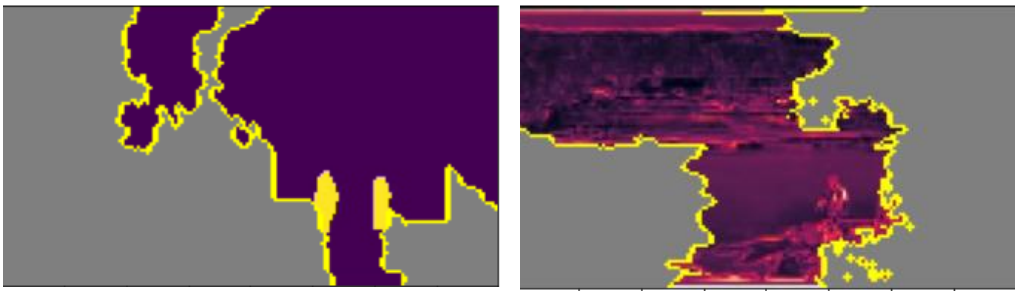
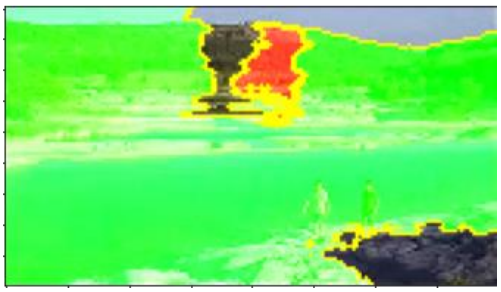


Figura 6.7: Regioni più esplicative per ogni versione di campo lungo

Partendo dalla classe del campo lungo, l'unica che ha un'alta percentuale di successo in classificazione su tutte le reti, vediamo che la regione dell'immagine che più influenza le predizioni delle reti è diversa per ogni modello. Questo accade perché ogni versione di immagine mette in risalto features diverse dell'inquadratura, ad esempio per il dato di image segmentation il set di super pixel più esplicativi giustamente sono quelli in viola, ovvero quelli che non mettono in risalto la sagoma umana, presente in questo campo lungo proprio per valutare la sua importanza nella classificazione, solitamente in giallo o in verde. Per le altre due reti, come si poteva già intuire, le regioni più caratterizzanti sono quelli contenenti gli elementi di background, soprattutto per il modello RGB che si serve anche dei colori, in questo caso del mare e della spiaggia, per individuare la classe corretta.
La tecnica di LIME permette anche di ottenere gli elementi di pro e di contro che la rete analizza per effettuare la predizione, naturalmente le regioni di pro, in verde, permettono la giusta classificazione e quelle di contro, in rosso, deviano il modello dalla scelta corretta (Fig. 6.7)



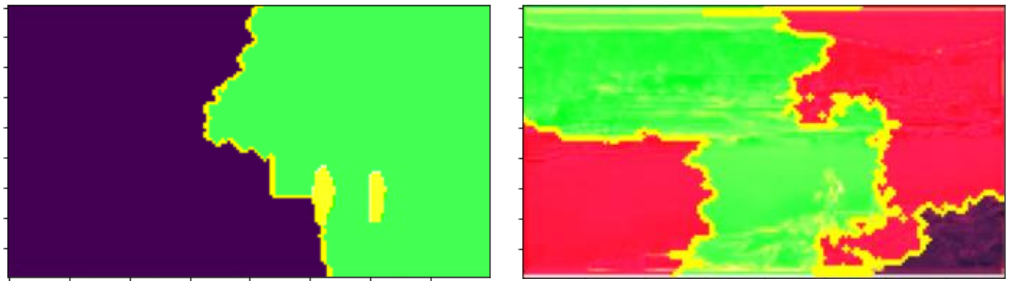


Figura 6.8: Esempio di esaltazione dei pros e cons di un immagine di campo lungo

Da queste rappresentazioni dei pro e dei contro notiamo ancora di più le differenze tra gli elementi caratterizzanti per ogni rete e come il modello di dati RGB abbia uno spettro di pros più ampio rispetto agli altri, coerente con l'abilità di questa? di classificare in modo più accurato di tutte questa tipologia di inquadratura.
Andando avanti nella carrellata di tipologie di scatto analizzate tramite explanable AI, osserviamo contemporaneamente le due inquadrature scelte che hanno la sagoma umana in risalto, il mezzo busto e il piano americano, che si trovano agli antipodi in merito al successo delle reti in fase di classificazione.

a)

b)

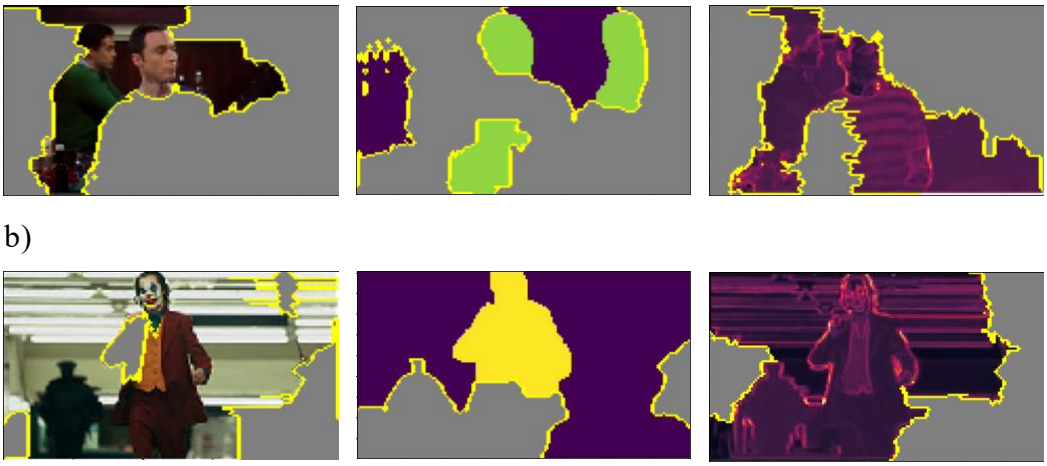
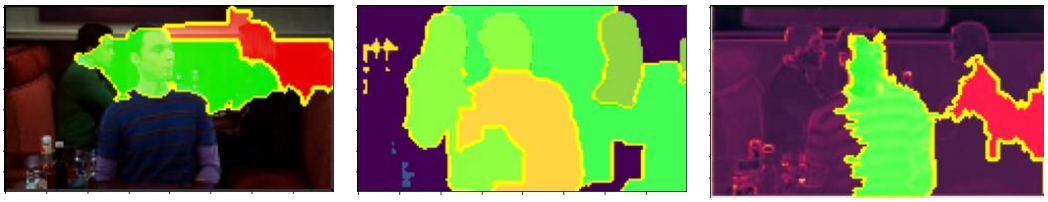


Figura 6.9: Regioni più esplicative per ogni versione di mezzo busto (a) e piano americano (b)

Osservando la Fig. 6.9, è possibile comprendere instantemente la motivazione per la quale le reti tendono a classificare meglio un mezzo busto rispetto ad un piano americano: la regione che viene presa in considerazione per quest'ultimo è molto più ampia e quindi dispersa rispetto a quella del mezzo busto. Per riconoscere queste due tipologie di inquadrature la sagoma umana e il punto in cui la macchina da

presa le taglia è fondamentale e quindi più la rete riesce ad individuare la persona all'interno dell'immagine e del rumore di fondo, più gli sarà facile classificare correttamente. Per questo motivo il mezzo busto, che ha un set di super pixel caratteristici concentrati per la maggior parte sulla figura umana, per tutte le reti, ha una percentuale di successo maggiore. Altra considerazione possibile rispetto alle tre reti è che esse prendono in considerazione parti diverse del corpo umano per comprendere davanti a quale tipologia di inquadratura si trovano. Ad esempio, alla rete VGG16 serve solo il volto umano ad una certa distanza dalla macchina da presa. Queste considerazioni sono evidenziate anche sulla versione pros e cons dei risultati in Fig. 6.10.

a)



b)

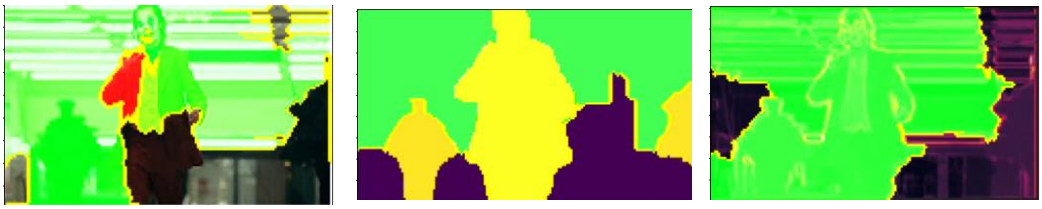


Figura 6.10: Esempio di evidenziamento dei pros e cons di un immagine di mezzo busto (a) e una piano americano (b)

In conclusione, analizziamo tramite LIME l'ultima inquadratura presa in considerazione perché tra le migliori classificate dal modello, il primissimo piano. Questa tipologia di scatto è diversa dalle altre perché non possiede né un background caratterizzante né una sagoma umana in parte o intera. Il primissimo piano è rappresentato solo dal volto del soggetto, spesso ad una vicinanza tale che il viso risulta anche tagliato e ciò rende tale inquadratura molto particolare e riconoscibile in alcune versioni dell'immagine (il primo piano è meno caratteristico perché spesso confondibile con un mezzo busto).

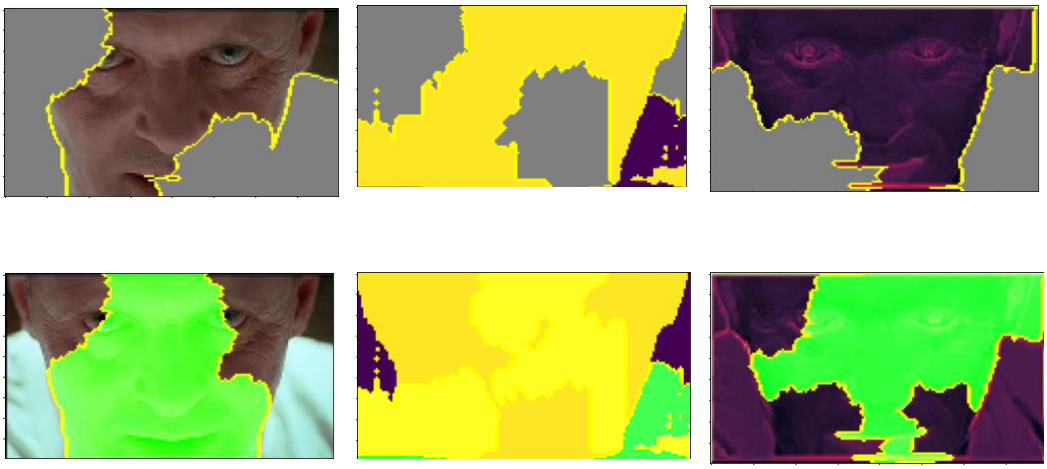


Figura 6.11: Analisi delle regioni di massima spiegabilità delle predizione per l'inquadratura primissimo piano

L'analisi di questa immagine è semplice perché il set di reti a disposizione trova nel volto dell'attore gli elementi principali per il riconoscimento dell'inquadratura ed è per questo che tale categoria è sempre ben classificata da tutte le reti proprio grazie all'estremità dello scatto.

CONCLUSIONI

Nel corso del presente lavoro è stato ampiamente evidenziato quale sia l'obiettivo desiderato: l'implementazione di una metodologia innovativa che fa uso di tecnologie di Deep Learning come Reti Neurali profonde, CNN, Ensemble Learning, da utilizzare nel campo cinematografico per classificare in modo accurato le inquadrature più adoperate nella movie industry (Campo Lungo, Campo Medio, Figura Intera, Mezzo Busto, Mezza Figura, Piano Americano, Primo Piano e Primissimo piano). L'intelligenza artificiale, madre di ogni algoritmo utilizzato, è risaputo essere uno strumento che aiuta ad ottimizzare compiti pratici ed effettuare task metodiche, per questo tale sperimentazione è stata ideata per testare la possibilità di applicare le tecniche del Deep Learning in un campo ancora agli albori della sperimentazione, come quello cinematografico, con lo scopo di creare in futuro possibili tool automatizzati a supporto degli addetti ai lavori su vari fronti. Come descritto nella sezione delle applicazioni pratiche, l'idea originale è quella di classificare set fotografici o riprese cinematografiche, che altro non sono che un insieme di fotogrammi e inquadrature, al fine di aiutare un fotografo o un addetto al video editing a "riordinare i pezzi" risparmiando molto tempo nella ricerca delle foto o dei video da inserire nella giusta sequenza. Per questi operatori, avere un'indicazione sui file che devono gestire è estremamente utile e permette di ottimizzare il lavoro. Difatti se i fotogrammi sono arricchiti del metadato rappresentante la tipologia di inquadratura, l'addetto non avrà necessità di vagliare tutti i file ma ricercherà solo all'interno della classe di scatti di suo interesse. Oltre a queste due applicazioni, vi sono altri campi in cui la classificazione degli scatti è molto utile come l'ottimizzazione della creazione di trailer (sempre nella medesima modalità del video editing) o la creazione di una smart reccommendation dei contenuti streaming. La classificazione portata avanti in questo elaborato potrebbe essere in grado, in sinergia con altri algoritmi di Deep Learning che svolgono detection intelligenti (colonne sonore o attori), di riconoscere il genere di un film, il contenuto e l'autore tramite le inquadrature più usate con cui poter arricchire i video con una serie di metadati che aiutano all'implementazione di una

raccomandazione intelligente dei contenuti di una piattaforma. La metodologia creata consiste in uno Stacking Ensemble Learning che sfrutta la conoscenza di potenti classificatori come le reti neurali convoluzionali profonde VGG16 migliorandone ancora ulteriormente le performance. Nella pratica questo modello, dopo l'addestramento, potrà ricevere i fotogrammi delle riprese effettuate in giornata dai registi e capire da essi se la scena contiene primi piani, campi lungi ecc. ma anche riconoscere l'autore, il genere, il contenuto e quindi aggiungere una serie di metadati preziosi al lavoro degli operatori. Ciò è permesso dalla stabilità di classificazione ottenuta, che non incorre in Overfitting come molti modelli presenti in letteratura, e dall'accuratezza raggiunta in fase di test (≥90%) che è già un primo ottimo traguardo e che spinge a continuare gli studi su tale fronte al fine di migliorarne la precisione. Il risultato ottenuto non è ancora tale da poter pensare di applicare lo strumento allo svolgimento delle task reali dell'industria cinematografica ma traccia la giusta direzione e permette di confermare che, tramite l'utilizzo di Big Database, composti da una grande quantità di immagini, per svolgere l'addestramento del modello (non esistendo set di training già costituiti, per questo approccio sono state utilizzati in totale 10.000 campioni etichettati a mano, poco in confronto ai reali dataset con cui vengono allenate le reti presenti allo stato dell'arte) è possibile aumentare le performance considerevolmente, raggiungendo l'obiettivo prefissato di un'accuratezza che rasenta il 100%.

Altra considerazione importante è che il training del modello è stato effettuato senza un monitoraggio della sequenza dei campioni forniti all'algoritmo e quindi la casualità dell'ordine delle immagini può migliorare o peggiorare le prestazioni in fase di addestramento dipendentemente dal set fornito. Un metodo implementabile per trovare il set più performante e migliorare quindi l'accuratezza è la *cross validation* (implementato tramite K-fold di sklearn). Questa divide il training set in un numero scelto dall'utente di K campioni tra addestramento e test, i quali saranno adoperati dal modello in ordine sparso durante la fase di training così da individuare il *best case* relativo ai set di dati. Altro approccio che si rimanda ad uno sviluppo futuro volto al *boosting* delle performance, è l'implementazione del Federated Learning, di cui è stata fornita un'infarinatura teorica in appendice, a valle delle tre reti VGG16 così da sfruttare i vantaggi da esso garantiti come la possibilità di utilizzare diversi dataset eterogenei provenienti da più fonti per il training del modello o l'ottimizzazione del carico computazionale.

APPENDICE

Federated Learning

Durante la parte di sperimentazione presente in questo elaborato è stato utilizzato un approccio di Ensemble Learning per raggiungere risultati accurati nella classificazione delle inquadrature cinematografiche ma, come già introdotto, un'altra tecnica innovativa e potenzialmente efficace per svolgere questo compito e ancora mai sperimentata nell'ambito creativo, è il Federated Learning. Questo approccio differente è inserito nell'appendice dell'elaborato al fine di fornire al lettore una conoscenza a tutto tondo, anche se esclusivamente teorica, delle possibili tecniche presenti in letteratura utilizzabili allo scopo della tesi.

L'apprendimento federato (o meglio conosciuto come Federated Learning, termine americano coniato da Google AI che ha introdotto la tecnica nel 2017) è una metodologia che, diversamente dalle tecniche tradizionali che usufruiscono di dati centralizzati su server i quali ospitano il modello previsionale, permette l'addestramento di modelli di machine learning o deep learning come le reti neurali, adoperando dati locali depositati su una moltitudine di nodi senza lo scambio degli stessi tra un nodo e l'altro o verso un modello centrale.

L'idea alla base di questa metodologia è spostare la fase di training del modello all'origine dei dati movimentando delle copie di questo verso le sorgenti e non i dati verso una struttura centralizzata.

In altre parole, il federated learning connette diversi nodi computazionali (sorgenti dati) in un sistema decentralizzato di apprendimento, consentendo il training di modelli locali, ospitati su questi nodi, con i dati ivi generati e, grazie all'interscambio dei parametri (e.g. variabili sinaptiche come i pesi e il bias di una rete neurale) con il server centrale, consente di costruire un modello globale.

Questa tecnica si differenzia dal più comune *apprendimento distribuito* perché con il primo è possibile dividere il carico computazionale derivante dall'addestramento di una rete che adopera un dataset unico, su più nodi, mentre il federated learning ha lo scopo di effettuare il training di più modelli identici, copie del modello centrale, su un set di dati eterogenei provenienti da fonti diverse (i dati utilizzati nell'approccio distribuito sono omogenei e di stesso volume su ogni nodo, cosa che non accade per la tecnica federata in cui i dataset possono essere di dimensioni molto diverse).
Nella sezione successiva di questo capitolo, andremo più nel dettaglio nel processo di funzionamento di questa nuova metodologia approfondendo i concetti appena introdotti.

Processo generale e tipologie di apprendimento federato

In un apparato di apprendimento federato, i diversi nodi collegati possiedono ciascuno una copia locale del modello centrale di apprendimento automatico che si vuole addestrare (i diversi nodi computazionali scaricheranno una copia non addestrata del modello direttamente dal server centrale).
Tali copie svolgono la tradizionale fase di training direttamente sul nodo attraverso i dati locali forniti dallo stesso (come già accennato, ogni nodo, oltre ad avere una potenza computazionale, è anche una sorgente di dati locali).
Alla conclusione di questi training in locale, i diversi valori sinaptici/pesi generati dai modelli addestrati sui dati eterogenei dei nodi sono inviati al nodo master, il quale aggrega questi parametri e aggiorna le metriche del modello globale.
Il processo appena descritto è un'operazione iterativa che si ripete un numero *n* di volte (*federated round*) fino al raggiungimento dell'accuratezza target desiderata.
L'apprendimento federato è suddiviso in due tipologie che si differenziano per lo più per la modalità di gestione e supervisione del processo di training e interscambio dei parametri.
Il primo approccio, chiamato centralizzato (o monopartico), è caratterizzato da un solo nodo master (un server centrale) a cui è affidato il coordinamento degli step procedurali del training federato e la gestione dei nodi appartenenti al sistema di apprendimento. Questa entità centrale è incaricata di selezionare le sorgenti dati che

parteciperanno al federated learning e di aggregare i parametri generati dal training locale delle copie del modello globale.

Il federated learning centralizzato, nonostante i limiti derivanti dal carico computazione che il server centrale deve sopportare (tutti i nodi invieranno i parametri ad una sola entità che dovrà gestirli), è il più utilizzato nell'ambito del learning collaborativo di più nodi con dati eterogenei.

Questo approccio è spesso suddiviso in quattro diversi passaggi, il primo step è caratterizzato dalla scelta di un modello di apprendimento automatico (e.g. rete neurale) da parte dell'entità centrale e inizialmente addestrato dallo stesso sulla base dei dati in suo possesso.

Nel passo successivo tale modello generico inizializzato, base dell'intero processo di federated learning, è trasmesso ai nodi appartenenti alla rete di learning che addestrano queste copie con i dati in locale da loro generati (e.g. la tecnica di apprendimento potrebbe essere quella del calcolo del gradiente discendente in caso di reti neurali), aggiustando i parametri in base a questi e imparando a migliorare le performance (step 3).

Nello step finale (step 4) tutte le entità partecipanti al federated learning inoltrano i risultati ottenuti (parametri generati) al server centrale che li immagazzina, li lavora (esegue un tipo di aggregazione derivante dall'algoritmo utilizzato per svolgere il federated learning) e aggiorna il modello globale con i nuovi parametri prima di ritrasmetterlo ai nodi nuovamente e ripetere il processo.

L'iterazione termina quando si raggiunge l'accuratezza desiderata o il numero massimo di federated round (esempio in fig. 3.7).

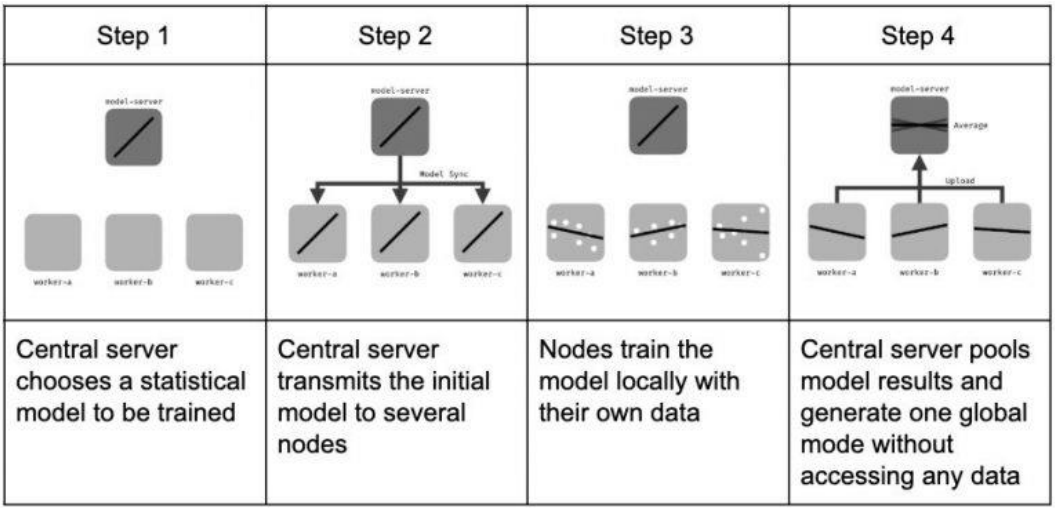


Figura 1[124]**:** Processo di Federated Learning in approccio centralizzato

L'altra tipologia di federated learning, spesso adoperata, è quella dell'apprendimento decentralizzato in cui i nodi che partecipano alla procedura sono in grado di autogestirsi e coordinarsi autonomamente (queste entità hanno la capacità di procurarsi il modello centrale e di scambiarsi i parametri in modo autonomo, per il resto il processo resta identico).

Questa tecnica risolve la questione di rallentamento del processo dovuto al *bottleneck* costituito dal nodo centrale e dal carico computazionale che esso si ritrova a gestire perché le entità partecipanti al federated learning hanno la capacità di scambiarsi i risultati e aggiornare il modello globale senza un nodo centrale che faccia da supervisor.

Anche questo approccio possiede dei limiti derivanti dalla configurazione della rete che può condizionare le performance di apprendimento.

Alla base del processo di apprendimento Federated troviamo due algoritmi di ottimizzazione principali introdotti entrambi da Google[125] nel 2017.

Il primo è il *Federated Stochastic Gradient Descent* (FedSGD), evoluzione del più noto algoritmo di discesa stocastica del gradiente (calcolo del gradiente negativo su un subset di dati casuali al fine di trovare la direzione giusta per raggiungere il minimo della funzione di costo generata dal training), utilizzatissimo dagli addetti ai lavori nell'ambito dell'apprendimento delle reti neurali artificiali.

[124] http://www.ihal.it/che-cose-lapprendimento-federato/

[125] McMahan, H. Brendan, Eider Moore, Daniel Ramage, Seth Hampson and Blaise Agüera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." AISTATS (2017).

Il FedSGD utilizza lo stesso criterio dell'algoritmo padre ma secondo un approccio federato e quindi viene implementato sull'intero bagaglio dati di un sottoinsieme di nodi partecipanti all'apprendimento federated e i gradienti calcolati sono i parametri scambiati con l'entità centrale e utilizzati per l'aggiornamento del modello globale.

L'algoritmo più famoso in ambito di apprendimento federato, cuore di questa metodologia, è il *Federated Averaging* (FedAVG). In fig. 3.8 possiamo vedere lo pseudo-codice che lo descrive in modo dettagliato.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Figura 2[126]: Pseudo-codice relativo al FedAVG

Il Federated Averaging è un espansione dell'algoritmo di gradiente stocastico in versione federata, appena descritto.

Questa tecnica permette alle entità partecipanti all'apprendimento federato di svolgere molteplici aggiornamenti e di trasmettere i pesi derivanti dalla fase di training dei vari modelli locali piuttosto che i gradienti.

Il federated averaging si caratterizza per il fatto che, in caso le entità della rete di apprendimento federato vengano inizializzate in modo identico, la media dei gradienti calcolati equivale alla media dei valori sinaptici.

[126] https://medium.com/datadriveninvestor/an-overview-of-federated-learning-8a1a62b0600d

Vantaggi derivanti dalla tecnica di Federated Learning

I vantaggi primari derivanti dall'approccio federato per il training di un modello sono diversi: in primo luogo esso rende possibile l'utilizzo di diversi dataset eterogenei per addestrare uno stesso modello a svolgere una determinata task in modo sempre più accurato e quindi di acquisire esperienza da una estesa varietà di set di dati (questo potrebbe portare un grande vantaggio nella classificazione delle inquadrature cinematografiche perché, come visto nella parte di trattamento dati e di analisi, già nell'attuale processo di ensemble learning sono stati utilizzati dataset differenti per addestrare uno stesso modello e quindi, in caso di sperimentazione dell'approccio federato, i risultati di tali training locali potrebbero essere agglomerati in un modello globale al fine di migliorare l'accuratezza).
Un altro vantaggio sostanziale deriva dal disporre di copie dei modelli sui vari nodi, il che permette di non dover effettuare uno scambio dei dati computazionalmente pesante e, in conseguenza, di ridurre la latenza del processo.

SITOGRAFIA

https://www.cospe.org/wpcontent/uploads/2019/07/03_dossier_INTELLIGENZA ARTIFICIALE_080719-1.pdf

www.morpheos.eu/2018/11/26/intelligenza-artificiale-forte-e-intelligenza-artificiale-debole
https://www.youtube.com/watch?v=Quh6x4kG6VY

https://youtu.be/lEfrr0Yr684

https://www.youtube.com/watch?v=e3Jy2vShroE

https://www.ai4business.it/intelligenza-artificiale/machine-learning/machine-learning-cosa-e-applicazioni/

https://www.filodiritto.com/machine-learning

https://www.beantech.it/blog/articoli/machine-learning-cosi-le-macchine-imparano-ad-migliori/
https://www.deeplearningitalia.com/a-general-introduction-to-learning-methods-2/
https://www.hudi.it/2019/10/02/i-tre-tipi-di-machine-learning/
https://www.4next.eu/blog/intelligenza-artificiale-definizione/
https://www.impresacity.it/approfondimenti/20434/i-modelli-del-machine-learning.html
https://it.mathworks.com/discovery/machine-learning.html

www.andreaminini.com/ai/machine-learning/ensemble-learning

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

https://www.ai4business.it/intelligenza-artificiale/deep-learning/

https://www.intelligenzaartificiale.it/intelligenza-artificiale-forte-e-debole/

https://www.intelligenzaartificiale.it/deep-learning/
https://it.mathworks.com/discovery/deep-learning.html

https://www.developersmaggioli.it/blog/il-transfer-learning/
https://www.ionos.it/digitalguide/

https://www.deeplearningitalia.com/una-panoramica-introduttiva-su-deep-learning-e-machine-learning/ https://www.dsi.unive.it/~srotabul/files/AppuntiReti Neurali.pdf

https://www.guru99.com/deep-learning-tutorial.html

https://www.asimovinstitute.org/neural-network-zoo/

https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

http://www.crescenziogallo.it/unifg/dottorato-medicina-traslazionale-2018/Reti%20neurali%20artificiali%20-%20Concetti%20base.pdf

https://www.youtube.com/watch?v=aircAruvnKk&t

https://medium.com/@ryaboug/neural-networks-deep-learning-teaching-a-computer-to-read-3e856e2d4a504

https://www.kaggle.com/c/dogs-vs-cats

https://deepai.org/dataset/mnist

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

https://www.youtube.com/watch?v=IHZwWFHWa-w&feature=youtu.be

https://www.youtube.com/watch?v=Ilg3gGewQ5U

https://www.youtube.com/watch?v=tIeHLnjs5U8

https://andreaprovino.it/gradient-descent-deep-learning-neural-network/

https://365datascience.com/sum-squares/

https://lorenzogovoni.com/algoritmo-discesa-del-gradiente/

https://www.retineuraliartificiali.net/algoritmo-di-backpropagation/

https://www.domsoria.com/2018/04/rete-neurale-feed-forward-e-back-propagation/

https://www.okpedia.it/algoritmo_back_propagation_reti_neurali

http://www.mtcube.com/SIS/Lezioni5_6_7.pdf

https://www.youtube.com/watch?v=jWT-AX9677k&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=9

https://youtu.be/DEMmkFC6IGM

http://bias.csr.unibo.it/maltoni/ml/DispensePDF/9_ML_DeepLearning.pdf

https://www.youtube.com/watch?v=5T-iXNNiwIs

https://lorenzogovoni.com/architettura-di-rete-neurale-convoluzionale/

https://youtu.be/YRhxdVk_sIs

https://youtu.be/cNBBNAxC8l4

https://youtu.be/qSTv_m-KFk0

https://youtu.be/ZjM_XQa5s6s

https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/

https://www.spindox.it/it/blog/reti-neurali-convoluzionali-il-deep-learning-ispirato-alla-corteccia-visiva/

https://www.developersmaggioli.it/blog/reti-convoluzionali/

https://tech.everyeye.it/articoli/speciale-cos-e-convolutional-deep-learning-cosa-serve-33066.html

https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

https://medium.com/@ODSC/what-is-federated-learning-99c7fc9bc4f5

https://medium.com/datadriveninvestor/an-overview-of-federated-learning-8a1a62b0600d

https://it.wikipedia.org/wiki/Apprendimento_federato

http://www.ihal.it/che-cose-lapprendimento-federato/

https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

https://www.dreamvideo.it/articoli/221/l-inquadratura-campi-e-piani

https://www.griffithduemila.com/art/tipi-di-inquadrature-cinematografiche-campi-e-piani.html

http://www.marellidudovich.gov.it/old/LEZIONI_AUDIOVIDEO/01_inquadratura.pdf

https://www.lascimmiapensa.com/2020/02/25/tipi-inquadrature-cinema/2/

https://cinemaescuola.files.wordpress.com/2016/09/scala-delle-inquadrature3.pdf

http://www.cineclub.bz.it/index.php/corso-di-video/377-inquadrature-e-movimenti-di-macchina

http://www.cinefile.biz/le-inquadrature-cinematografiche

http://cineclick.utetuniversita.it/linguaggio/inquadratura/piano-americano-/

https://www.telegraph.co.uk/films/0/irishman-review-netflix-robert-de-niro-sensational-history/

https://creativefuture.co/artificial-intelligence-automation-film-video-machine-learning-editing-robots-cinematography/

https://www.tomshw.it/altro/film-e-trailer-tutti-uguali-con-lintelligenza-artificiale-potrebbe-andare-peggio/

https://www.cinescuola.it/pianosequenza/

http://polisemantica.blogspot.com/2016/04/il-primo-piano-linquadratura-prediletta.html

https://lorenzogovoni.com/support-vector-machine/

https://it.mathworks.com/matlabcentral/fileexchange/68383-ensemble-learning-toolbox

https://en.wikipedia.org/wiki/Ensemble_learning#Stacking

https://mc.ai/icu-survival-prediction-using-ensemble-learning-stacking/

https://blog.statsbot.co/ensemble-learning-d1dcd548e936

https://neurohive.io/en/popular-networks/vgg16/

https://it.wikipedia.org/wiki/RGB

https://supervise.ly/deep-lab-v-3-plus/overview

https://www.jeremyjordan.me/semantic-segmentation/

https://it.qwe.wiki/wiki/Aspect_ratio_(image)

https://www.xn--photocaf-80a.it/bilanciamento-del-bianco/

https://www.kaggle.com/

https://opencv-python-tutroals.readthedocs.io/py_tutorials/py_video_display.html

https://pythonhosted.org/Pafy/

https://stackoverflow.com/automatic-white-balancing-with-grayworld-assumption

https://machinelearningknowledge.ai/opencv-tutorial-image-colorspace-conversion-using-cv2-cvtcolor

https://it.wikipedia.org/wiki/Spazio_colore_Lab

https://www.jeremyjordan.me/semantic-segmentation/

https://towardsdatascience.com/semantic-image-segmentation

https://medium.com/analytics-vidhya/semantic-segmentation-in-pspnet

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

https://www.cityscapes-dataset.com/dataset-overview/

https://github.com/divamgupta/image-segmentation-keras

https://github.com/kairess/human_segmentation

https://github.com/bonlime/keras-deeplab-v3-plus

https://towardsdatascience.com/review-hypercolumn-instance-segmentation

https://blog.christianperone.com/2016/01/convolutional-hypercolumns-in-python/

https://www.kaggle.com/rupeshs/hypercolumn-test-vgg16

https://towardsdatascience.com/step-by-step-vgg16

https://www.geeksforgeeks.org/vgg-16-cnn-model/

https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/

https://devacademy.it/json/

https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/

https://en.wikipedia.org/wiki/Confusion_matrix

https://medium.com/@darshita1405/superpixels-and-slic

https://towardsdatascience.com/interpretable-machine-learning-for-image-classification-with-lime

https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python

https://towardsdatascience.com/k-means-and-pca-for-image-clustering-a-visual-analysis

https://it.wikipedia.org/wiki/Analisi_delle_componenti_principali

https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means

https://medium.com/pursuitnotes/k-means-clustering-model-in-6-steps

BIBLIOGRAFIA

[1] E. Brynjolfsson e A. McAfee, *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies,* 18 febbraio 2014

[2] R. Marmo, *Algoritmi per l'intelligenza artificiale (versione in eBook) Capitolo 1 - Progettazione dell'algoritmo - Dati e Machine Learning - Neural Network - Deep Learning*

[3] J. R. Searle, *Argomento della Stanza Cinese, Minds, Brains and Programs),* rivista Behavioral and Brain Science, 1980

[4] E.Yudkowsky, *ArtificialIntelligence as a Positive and Negative Factor in Global Risk,* MachineIntelligenceResearchInstitute, 2008

[5] N. Boldrin, *AI Artificial Intelligence: Come è nata, come funziona e come l'Intelligenza,* 2018

[6] C.M. Bishop, *Pattern Recognition and Machine Learning, Springer, New York, NY,* Springer, 2006. - 738 p

[7] A. Geitgey*, "Machine Learning is Fun!"*, Maggio 2014

[8] C. Shao*, A Quantum Model for Multilayer Perceptron*, arXiv: Quantum Physics, 2018

[9] McMahan, H. Brendan, Eider Moore, Daniel Ramage, Seth Hampson and Blaise Agüera y Arcas. "*Communication-Efficient Learning of Deep Networks from Decentralized Data*." AISTATS (2017)

[9] A. Giaime Alonge, *"Il cinema. Tecnica e linguaggio. Un'introduzione",* Torino, Kaplan, 2011

[10] L. Canini, S. Benini, and Ri. Leonardi. "*Classifying cinematographic shot types"*. Multimedia Tools and Applications, 62:51–73, 2011

[11] S. Benini& L. Canini, & R. Leonardi. *"Estimating cinematographic scene depth in movie shots".* 2010 IEEE International Conference on Multimedia and Expo, ICME 2010. 855 - 860. 10.1109/ICME.2010.5582611

[12] M. Svanera et al. *"Who Is the Film's Director? Authorship Recognition Based on Shot Features."* IEEE MultiMedia 26.4 (2019): 43–54. Crossref. Web

[13] D. Ciresan, U. Meier, and J. Schmidhuber. *Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition* (CVPR), 2012 IEEE Conference on, pages 3642–3649. IEEE, 2012.

[14] D. C Cireşan, U. Meier, J. Masci, L. M Gambardella, and J. Schmidhuber. *High-performance neural networks for visual object classification*. arXiv preprint arXiv:1102.0183, 2011.

[15] S. Arora, A. Bhaskara, R. Ge, and T. Ma. *Provable bounds for learning some deep representations.* arXiv preprint arXiv:1310.6343, 2013.

[16] Hariharan, Bharath et al. "*Hypercolumns for Object Segmentation and Fine-Grained Localization.*" 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)

[17] Tan, Hui Fen, K. Song, Madeilene Udell, Yiming Sun and Yujia Zhang. *""Why Should You Trust My Explanation?" Understanding Uncertainty in LIME Explanations."* arXiv: Learning (2019)

RINGRAZIAMENTI

Le puntante di un cammino universitario sono tante e tutte da 80 minuti, un po' come le grandi serie TV. Le mie hanno avuto tanti interpreti per me fondamentali e quindi, giunto ai titoli di coda della mia serie "Universitaria", con le ultime battute di questa tesi sento la necessità di rivolgere un ringraziamento a chi è stato attore, regista, produttore, co-protagonista, partner o anche solo una comparsa del percorso che mi ha visto protagonista.

In ordine sparso; Ringrazio la produzione, la mia famiglia: non comune, particolare, allargata, moderna. Loro sono sempre stati lì, punto fermo di ogni mio momento quotidiano o straordinario, pronti sempre a sostenermi durante le mie cadute e le mie crisi, tante, e a festeggiare le mie prime gioie, il primo esame superato, i primi traguardi, fino a questo ultimo finale. In particolare, sento di dover ringraziare i miei genitori, veri produttori di questo piccolo successo, che hanno sempre fatto molto di più di quello che un figlio possa mai desiderare.

Ringrazio la regia, professori e assistenti, che in questo percorso a puntante mi hanno permesso di crescere e raggiungere una maturità professionale che ora mi permette di avere un ruolo importante anche su un nuovo palcoscenico come il mondo del lavoro. Particolarmente, rivolgo la mia gratitudine alla mia relatrice Tania Cerquitelli, che mi ha permesso di svolgere la tesi su un argomento a me molto caro come il cinema e che è stata sempre molto disponibile al dialogo volto a migliorare giorno dopo giorno questo elaborato, e al dottorando Bartolomeo Vacchetti, angelo custode di queste battute finali, che mi ha seguito e supportato in in ogni mio passo, fin dalle basi.

Ringrazio i co-protagonisti, i miei fratelli e gli amici di una vita. Il divertimento e le risa con loro non sono mai mancati, neanche nei momenti difficili, da loro ho imparato che pochi e importanti è meglio di tanti (e non parlo di esami!!!)

Ringrazio la partner della mia personale commedia romantica, la mia ragazza, unica e sola, che paziente mi ha sostenuto e aiutato ad affrontare ogni frangente di questo ultimo duro periodo, regalandomi ogni fine settimana un sorriso, un abbraccio e un bacio anche quando la mia testa era da un'altra parte e le mie

attenzioni non erano all'altezza, grazie! (Si dice: "dietro ad un grande uomo, c'è una grande donna", io direi: "dietro ad un grande traguardo, c'è sempre una donna meravigliosa").

Ringrazio tutti gli altri attori, nuovi e vecchi amici, colleghi, che hanno caratterizzato questi lunghi anni e mi hanno permesso di intrecciare la mia trama con le loro e condividere il palcoscenico anche solo per una volta.

In aggiunta, sento una forte gratitudine anche verso le tante comparse, ognuna importante in modo diverso, le quali hanno influenzato anche solo in piccolo il mio modo di interpretare il resto del copione, vi ringrazio!

In conclusione, ringrazio il set, Torino e il suo Politecnico, croce e delizia del mio passato e del mio avvenire.