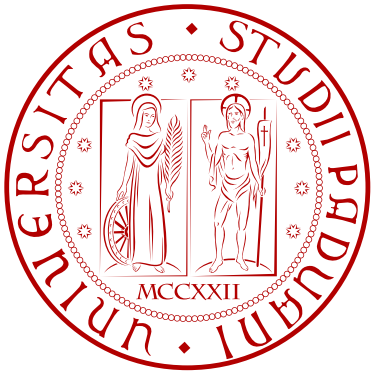


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Tesi di Laurea Magistrale in Ingegneria Informatica

RICONOSCIMENTO E CLASSIFICAZIONE DI DISCONTINUITÀ IN NASTRI MAGNETICI AUDIO MEDIANTE RETI NEURALI E TECNICHE DI VISIONE COMPUTAZIONALE: UN SOFTWARE INNOVATIVO BASATO SU UN ESTESO INSIEME DI ADDESTRAMENTO

Laureando: Matteo Pignotti

Relatore: Carlo Fantozzi

Correlatore: Niccolò Pretto, Sergio Canazza

Data Laurea: 24/02/2020

Anno Accademico 2019/2020

Sommario

Applicativi software possono offrire un valido supporto ai tecnici audio e musicologi nelle procedure di conservazione di un documento sonoro nella sua interezza, agevolandoli nelle fasi più ripetitive e inclini a errori. In questo caso vengono riportate e spiegate alcune tecniche di individuazione e classificazione di discontinuità che possono essere presenti nel nastri considerati, che necessitano essere rilevate e documentate per avere una descrizione totale di tali documenti. L'estrazione di queste discontinuità, al mio arrivo, avveniva tramite uno script che analizzava la media delle variazioni di colore dei pixel della regione del video preso in esame, e se questa superava una certa soglia allora il frame poteva essere considerato una discontinuità. L'insieme di queste rilevazioni venivano poi usate per allenare una rete neurale con il fine di riuscire a classificare tali frame estratti. Come conseguenza ai lavori precedenti e alle problematiche riscontrate (dataset sporco, rilevazione multipla di una stessa discontinuità, rete neurale eccessivamente grande per la classificazione), con questo elaborato si cerca di capire se un'eventuale riduzione di dimensionalità delle reti neurali analizzate precedentemente possa fare al caso nostro, verificando anche se applicando Transfer Learning, si poteva avere delle buone performance senza dover allenare una nuova rete nella sua totalità. In questo elaborato verranno poi analizzati i vari risultati ottenuti.

Una volta fatto questo, è stato pensato di ridurre gli errori e i rumori all'interno dei frame estratti, quindi è stata idealizzata una nuova tecnica di individuazione e estrazione. Essa si basa inizialmente nell'indicare manualmente l'istante temporale e la tipologia di discontinuità rilevata su un opportuno file CSV, per poi estrarre tali frame tramite due script. Il primo ha il compito di dare all'utente la possibilità di individuare in maniera grossolana la regione di interesse nella quale si vuole osservare la presenza o meno della discontinuità, e di estrarre autonomamente i frame seguendo l'opportuno file CSV. Il secondo invece dà la possibilità all'utente di poter scegliere il frame più consono a far parte del dataset e quindi riducendo al minimo rumori ed errori di duplicazione. E' stato poi dimostrato come una semplice concatenazione di pochi layer convoluzionali potesse fare a al caso nostro rendendo la classificazione semplice e veloce.

Ringraziamenti

Mi è doveroso dedicare questo spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione dello stesso. Desidero ringraziare innanzitutto il professor Fantozzi e il professor Canazza che grazie alla loro collaborazione sono riusciti a venirmi incontro in un momento del mio percorso abbastanza complicato, dato l'improvviso cambio di relatore. Ringrazio l'intero gruppo del CSC con i quali ho condiviso pranzi, caffè e risate, i quali hanno creato un bel ricordo di queste mia esperienza all'interno del complesso. Un ringraziamento particolare va a Niccolò Pretto, mio correlatore, che con i suo consigli, la sua instancabile presenza, e la sua meticolosità, mi ha aiutato al raggiungimento del mio risultato cercando di ambire sempre al meglio del mio progetto.

Questi cinque anni mi hanno veramente cambiato; conoscenze, amici, compagni di corso mi hanno formato, e mi hanno reso ciò che sono ora.

In particolare, voglio dire grazie di cuore ad Elisabetta, che in questo ultimo periodo si è subita ansie, crisi di nervi, riuscendo a tenermi in piedi nei momenti in cui vacillavo e pensavo di non farcela. Spero di poter continuare a contare su di lei anche in futuro.

Un ringraziamento speciale va anche ai miei genitori, che hanno sempre creduto in me, appoggiando le mie scelte e spronandomi ogni giorno a dare il meglio. Voglio spendere una parola anche per Daniele, Massi, Teo, Masi, Gio, Bore i quali mi sono stati accanto nelle varie difficoltà universitarie e compagni di pomeriggi interminabili in Ge; *"evergreen"*.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La musica è sempre stata uno dei pilastri cardine dell'espressivitá dell'uomo. Per sua natura, essa è intangibile e rientra quindi nella categoria degli ***intangible cultural heritage*** che l'UNESCO definisce come l'insieme delle pratiche, delle rappresentazioni, delle espressioni, e della conoscenza che comunità, gruppi e, in alcuni casi, individui riconoscono come parte del patrimonio culturale. Detto questo, gli oggetti e i supporti relativi all'esecuzione musicale possono essere considerati parte di questo patrimonio culturale immateriale (spartiti, strumenti musicali, costumi, poster, registrazioni audio). Tale patrimonio viene racchiuso sotto il termine ***musical cultural heritage*** [17,20]

Col passare degli anni, l'innovazione tecnologica ha permesso all'uomo di usare nuovi strumenti capaci di registrare e riprodurre sempre più fedelmente questo patrimonio musicale. Grazie a strumenti di registrazione sempre più all'avanguardia e affidabili, si è dato vita ad un vasto e vario patrimonio di grande interesse storico e scientifico (interviste, reportage, cronache, concerti o sessioni musicali). Tutto questo crea una ricca raccolta che necessita di essere preservata e tramandata nel tempo, come testimonianza diretta di tempi ormai passati.

Secondo l'UNESCO, tuttavia, tale patrimonio potrebbe essere perduto. Le cause sono tra le più varie:

- Continua evoluzione tecnologica: i vecchi supporti di registrazione e la strumentazione ad essa collegate sono vittime dell'obsolescienza.
- La diminuzione dei pezzi di ricambio disponibili sul mercato per le attrezzature.
- Supporti fisici vulnerabili, soggetti ad usura e degrado nel tempo, con conseguente rischio di perdita del contenuto registrato.

Quindi, definite le problematiche, è d'obbligo trovare un protocollo adatto ed efficace alla conservazione di questi materiali e documenti. L'idea principale non è solo quella di conservare tali reperti in maniera tale da prevenire l'usura e il decadimento nel tempo, ma anche di attuare un trasferimento dell'informazione, riportando il contenuto in supporti sempre più all'avanguardia. Un esempio di questo trasferimento può essere il processo di *rimediazione* usato anche presso il Centro di Sonologia Computazionale (CSC) dell'Università degli Studi di Padova [1].

1.1 Il processo di rimediazione

Nel campo della conservazione è appropriato distinguere tra processo di *digitalizzazione* di un reperto musicale e processo di *rimediazione* [1]. Il primo è inteso come creazione di una copia in formato digitale, mentre il secondo è finalizzato alla conservazione di un documento sonoro nella sua interezza il quale deve seguire una serie di procedure e regole ben definite.

Al *CSC* l'intero processo di rimediazione può essere diviso in tre fasi ben distinte: *preparazione del supporto*, *trasferimento del segnale* ed *elaborazione e archiviazione dei dati*.

La rimediazione quindi non consiste solo in un trasferimento del contenuto del supporto originale in un altro ma, con il restauro, è necessaria la creazione di una documentazione dettagliata la quale deve seguire determinati standard [1]. L'originalità e l'integrità del documento devono essere gli obiettivi che tale processo deve raggiungere. Per questo, oltre all'acquisizione dell'audio, viene anche creato un insieme di file multimediali (foto, video) relativi ai documenti cartacei correlati (copertine, foto, libretti d'opera).

1.2 Copie conservative

Il risultato del processo di rimediazione è la cosiddetta *copia conservativa*. Essa rappresenta un gruppo organizzato di dati che identifica tutta l'informazione portata del documento originale nella sua totalità, in aggiunta alla sua descrizione e alla documentazione relativa al processo di conservazione. Dato che la copia conservativa è finalizzata alla conservazione a lungo termine, questa deve descrivere il documento originale nel minimo dettaglio [18]. Estremamente importante è avere una sincronizzazione tra documentazione video e l'audio poichè, l'insieme delle due cose, riporta informazioni sullo stato fisico del supporto e sull'eventuale presenza di alterazioni al momento della digitalizzazione.

Come vedremo in seguito, il video assume un'importanza primaria quando si tratta di copie conservative di nastri magnetici. Il video, a differenza dell'audio, può essere codificato in formato *lossy* (comprimere i dati attraverso un processo con perdita d'informazione che sfrutta le ridondanze nell'utilizzo dei dati), ovviamente prestando attenzione ad avere una risoluzione e un *bitrate* utili.

La struttura della copia conservativa è presentata in figura 1.1; essa deve contenere gli elementi di seguito riportati.

- File audio, in formato *lossless* BWF (Broadcast Wave Format), campionato ad almeno 96kHz e 24bit di risoluzione. Quando si parla di risoluzione si intende il numero di bit utilizzati per rappresentare i campioni. Solitamente si utilizzano 16 o 24 bit per campione, in base alla scelta si va ad aumentare rispettivamente la qualità del segnale. Anche la frequenza di campionamento merita opportuna attenzione; segnali analogici diversi possono dare luogo allo stesso segnale campionato (problema dell'*aliasing* risolvibile con il teorema del campionamento di *Nyquist-Shannon*).

- Documentazione multimediale, composta da foto del supporto originale e video acquisito durante la riproduzione.

- Metadati dei diversi file (*checksum*, specifiche tecniche dei diversi formati).
- Report descrittivo dell'intera copia conservativa e documentazione relativa al processo utilizzato per la rimediazione.

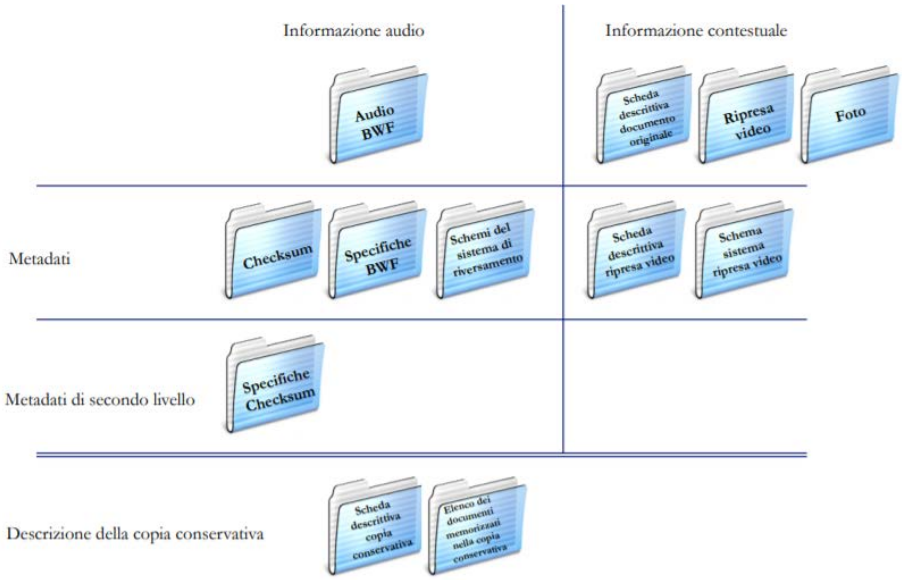


Figura 1.1: Struttura copia conservativa [1]

Presso il CSC i video vengono acquisiti con una videocamera professionale con una risoluzione di 720x576 pixel. L'utilizzo di formati e codifiche ad alta qualità però aumentano ovviamente le dimensioni della copia conservativa, rendendone difficile la condivisione, in particolar modo se richiesta via web. Importante è ricordare che l'eventuale accessibilità da parte di un pubblico esteso al documento potrebbe mettere a repentaglio l'integrità della copia stessa. Per questa ragione viene creata anche la *copia d'accesso*. Quest'ultima può presentare una qualità sonora inferiore ma, oltre a essere disponibile per la consultazione, permette al musicologo di testare interventi di restauro o sperimentare analisi di vario genere, impossibili da effettuare sulla copia di partenza senza il rischio di comprometterla.

1.3 Conservazione

Nel libro di Van Peursen viene detto [21]:

[...] the creation of digital objects − be it images of inscriptions or manuscripts, electronic versions of ancient corpora, or collections of secondary literature − is a crucial part of humanities research. It is more than just preparation for research.

La conservazione di opere e reperti può essere essenzialmente di due tipi: *attiva* o *passiva*. La prima richiede il trasferimento del contenuto originale in un nuovo supporto; l'idea principale è quella di preservare l'informazione nel tempo, e non il mezzo di trasissione. Questo processo non è reversibile e oltre a salvare il contenuto sonoro, protegge il supporto originale. Oltre a introdurre un maggiore grado di longevità al documento, questa tecnica dà la possibilità al pubblico di usufruire di opere prima difficilmente o parzialmente disponibili, in modo tale da facilitarne l'accesso.

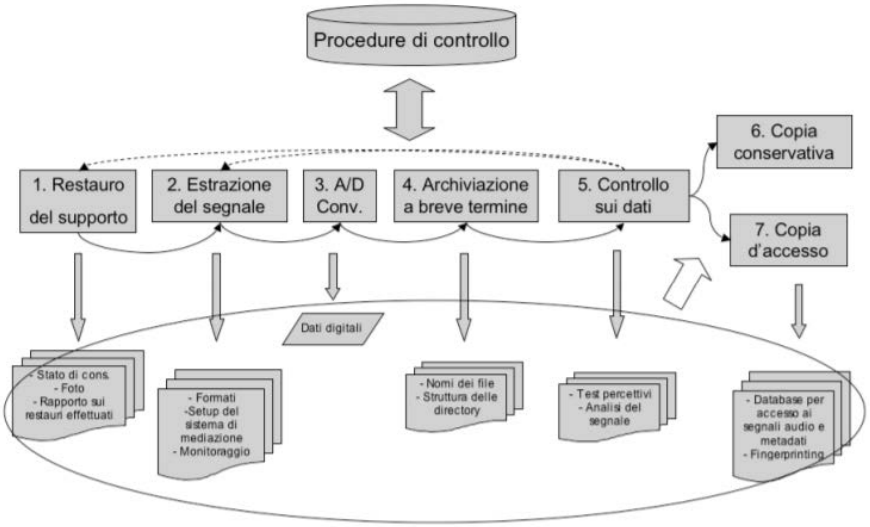


Figura 1.2: Struttura conservazione attiva [1]

La conservazione passiva invece coincide con tutti quei casi in cui la degradazione fisica del supporto può essere rallentata con una corretta conservazione, ma non fermata. Questo processo poi si divide in ulteriori due categorie:

- **Conservazione passiva diretta:** si agisce sull'ambiente in cui è conservato il documento sonoro (*condizioni di conservazione, procedure di manipolazione, etc.*).
- **Conservazione passiva indiretta:** si agisce sul supporto con trattamenti atti a stabilizzarne le condizioni fisiche senza alterarne la struttura e la composizione.

1.4 Digitalizzazione

Come detto in precedenza, rimediazione e digitalizzazione sono due processi diversi. Quest'ultima infatti è, secondo Smith (1999)[22],

it is primarily a method of providing access to rare, endangered, or distance materials [...] not permanent solution for preservation.

Questo processo ha preso sempre più piede con l'avvento dell'era digitale. Con questo termine si indica l'insieme di fasi atte alla conservazione di un segnale dal dominio dei valori continui (segnale analogico) a quello dei valori discreti (segnale digitale). Questo introduce maggiore longevità al documento, migliorandone e rendendone più facile la copia in supporti diversi, rendendolo più reperibile. Di conseguenza, grazie a questa tecnica, si rendono accessibili a un pubblico più ampio quei reperti sonori precedentemente impossibili da consultare nella forma originale.

Tutto questo però non è privo di aspetti negativi. Come primo problema si deve considerare il fatto che sono necessarie competenze specifiche per trattare il supporto nella maniera più corretta, non chè tecniche di standardizzazione per il raggiungimento di risultati di qualità e accettabili per l'uso archivistico. In secondo luogo è necessario disporre di tecnologie di archiviazione all'avanguardia e affidabili.

L'esportazione del segnale attuando la compressione, può provocare problemi nell'integrità del documento stesso. Il suono è un segnale continuo e per essere memorizzato deve essere campionato ottenendo così un segnale digitale. Tre sono i parametri che caratterizzano il campionamento; uno influenza lo spazio occupato, come ad esempio *il numero di canali*, gli altri invece la qualità del suono ovvero la *risoluzione* e la *frequenza di campionamento*.

1.5 Caso di studio: nastri magnetici

Il nastro magnetico è un supporto audio a memorizzazione magnetica, composto da una sottile fettuccia in plastica, rivestita a sua volta di un materiale sensibile alla polarizzazione. Il primo nastro magnetico di registrazione per uso audio fu prodotto dalla BASF AG nel 1935 destinato al primo registratore a bobine: il Telefunken Magnetophon k1. Questi registratori nacquero dall'idea base del registratore a filo, uno strumento in grado di salvare suoni su un filo d'acciaio.

Tutti i nastri registrabili sono prodotti ugualmente, utilizzando lo stesso tipo di tecnologia anche se usati in ambiti diversi; siano essi utilizzati per il settore video o per l'audio.

La storia dei nastri utilizzati per la registrazione e la riproduzione audio, è segnata da molte innovazioni tecnologiche rivoluzionarie, che hanno portato un enorme sviluppo durante i decenni: dalle bobine, alla musicassetta. Il segnale registratovi è prevalentemente analogico, anche se durante gli anni sono stati sviluppati metodi di registrazione digitale (DAT, Digital Audio Tape). Nel primo caso, gli interventi di scrittura vengono attuati tramite una testina in grado di registrare sul nastro per mezzo di un campo magnetico, generato da impulsi elettrici creati da una fonte esterna (es. microfono); la magnetizzazione del supporto rende l'informazione duratura

nel tempo. In fase di lettura la stessa testina legge questi campi magnetici traducendoli in segnali elettrici, attuando il procedimento inverso.

Figura 1.3: Esempio di magnetofono [4]

L'informazione sui nastri digitali, invece, a differenza di quelli analogici, è registrata tramite un segnale discreto; il segnale analogico viene campionato, tradotto come flusso di bit, per poi essere scritto sul mezzo di trasporto.

I nastri magnetici hanno una bassa aspettativa di vita; le cause più ricorrenti del loro deterioramento sono riportate di seguito:

- **Umidità**: l'acqua e la condensa a contatto con la superficie del nastro provocano con alta probabilità l'idrolisi, una reazione chimica che provoca l'alterazione della composizione del supporto e ne compromette l'uso favorendo inoltre la proliferazione di funghi e muffe.
- **Temperatura**: gli ambienti dove sono presenti grandi sbalzi di temperatura possono alterare la composizione chimica o deformare la struttura del nastro; temperature elevate possono infatti favorire e accelerare reazioni come l'idrolisi spiegata prima.
- **Campi magnetici**: la presenza di tali campi, generati per esempio da attrezzature come i microfoni, possono compromettere anche in maniera irreversibile i supporti magnetici.
- **Danni meccanici**: se non maneggiati con cura i nastri magnetici sono soggetti a danni fisici, come distorsioni o stropicciamenti. Tali danni inficiano la riproduzione solo localmente nel punto rovinato.
- **Polvere e impurità**: esse possono rappresentare fisicamente un ostacolo per eventuali testine di lettura, che non riescono ad avere un contatto preciso con la superficie del supporto alterandone la riproduzione.

I nastri magnetici sono tipicamente raccolti all'interno di cassette o bobine. In fase di riproduzione la bobina del nastro viene inserita in un mozzo (*hub*), mentre l'estremità libera del nastro viene fatta passare attraverso le testine di registrazione. Essa viene infine agganciata e raccolta da un'altra bobina uguale ma vuota.

Questi strumenti di avvolgimento possono presentarsi in formati diversi: possono essere con flangia o meno, con diametri diversi, oppure possono variare per il tipo di *hub*. Anche il nastro magnetico si può presentare in diverse configurazioni. La larghezza ad esempio può essere variabile:

- nastri da 2 pollici, per utilizzi professionali;
- nastri da 1 pollice;
- nastri da 1/2 pollice;
- nastri da 1/4 di pollice, il formato più diffuso;
- nastri da 1/8 di pollice, per le audio cassette.

E' utile fare ulteriori diverse distinzioni riguardo i nastri. Essi possono essere *magnetici* o *leader*. Sui primi vengono effettivamente registrati i suoni, mentre i secondi vengono usati alle estremità dei nastri stessi per essere agganciati alla bobina. Quest'ultimi non sono magnetizzabili, non vi si può registrare nulla e, a livello sonoro in fase di riproduzione, si percepisce una pausa. Questi due nastri sono collegati tra loro tramite una giunta, ovvero un'unione realizzata tramite nastro adesivo (fig. 1.4).

Figura 1.4: Esempio giunta su nastro magnetico [4]

Il numero di tracce, ovvero le suddivisioni longitudinali del nastro, può variare di volta in volta a seconda della larghezza del nastro. Nella maggior parte dei casi il nastro supporta due o quattro tracce, ma esistono situazioni in cui possiamo trovarne 8, 16 o 24. Il nastro a due tracce contiene nella maggior parte dei casi un segnale stereo. Quello a quattro invece, può contenere

due segnali stereofonici oppure quattro segnali mono indipendenti (quattro tracce registrate nello stesso senso, quadrifonia).

In fase di acquisizione dell'audio per la digitalizzazione dei nastri magnetici è necessario prestare attenzione a come vengono inizializzati alcuni parametri di lettura, per non inficiare la qualità della copia conservativa.

- L'equalizzazione: rappresenta il procedimento di filtraggio a cui è sottoposto un segnale audio per variarne il contenuto timbrico. La scelta di una curva errata distorce il segnale, producendo un risultato parzialmente o totalmente diverso e non riconducibile all'originale.

- La riduzione del rumore: i sistemi di registrazione possono introdurre del rumore derivante dal sistema stesso durante l'esecuzione.

- La velocità scorrimento del nastro: incide sulla qualità della registrazione. A una velocità bassa, infatti, corrisponde una limitazione nella gamma di frequenze correttamente registrabili. Ci sono sei standard di velocità ben distinti: 30 ips (76.2 cm/s), 15 ips (38.1 cm/s), 7.5 ips (19.05 cm/s), 3.75 ips (4.76 cm/s), 1.875 ips (2.38). Dato che un registratore che lavori a tutte queste velocità nella stessa macchina non esiste, le più comunemente usate sono 7.5 e 15 ips.

1.6 Discontinuità del nastro

Come già detto in precedenza, i nastri magnetici sono soggetti a degradazioni fisiche. Spesso, queste, sono riscontrabili ad una prima ispezione visiva del supporto e devono essere sempre documentate nella copia conservativa tramite fotografia. Il termine ***discontinuità*** sarà usato per indicare tutte quelle alterazioni presenti nel mezzo di trasporto del file audio nel suo stato di fabbricazione originale, rilevabile durante lo scorrimento del nastro. A volte i produttori di nastri stampano il proprio marchio o logo sul retro del nastro stesso. Queste marche saranno considerate come discontinuità, anche se non sono effettive alterazioni.

Diventa necessario quindi avere strumenti efficaci e automatici per rilevare tali imperfezioni; in questo modo i tecnici audio non dovranno scorrere l'interno nastro alla ricerca di tutte le discontinuità ogni qual volta analizzeranno il video, il musicologo invece potrà capire se tale suono, pausa, alterazione è voluta e registrata nel nastro oppure si tratta di un difetto di quest'ultimo. Le principali discontinuità riscontrabili durante la riproduzione del nastro sono riportate di seguito:

- Presenza di danni o rotture del nastro.

- Presenza di una o più pieghe o di increspature.

- Presenza di sporco, muffa, impurità, polvere o cristalli, o sostanze collose sul nastro, che accumulano sporco, muffa sulle testine e sulle guide del magnetofono durante la riproduzione.

- Presenza di giunte e/o segni, o marche sul nastro o sulle giunte stesse.
- Presenza di adesione tra due o più strati.
- Perdita del rivestimento magnetico. Le particelle stimolabili magneticamente possono disperdersi e accumularsi sul retro del nastro o peggio depositarsi sulle testine del magnetofono. Tali impurità possono compromettere la qualità della riproduzione, poichè impediscono la corretta lettura.
- Fenomeno del *cupping* per cui il bordo del nastro non è piatto rispetto al centro e si innalza sopra la testina durante la riproduzione.

Sia la creazione, che lo studio della copia conservativa, condividono un problema comune: l'errore umano. Visionare il video del nastro che scorre con il compito di rilevare la possibile presenza di discontinuità, è un lavoro che richiede una costante attenzione [19]. Dopo ore, il tecnico o musicologo potrebbe non notare alcuni dettagli e il lavoro finale potrebbe presentare alcune imperfezioni. Per tale motivo, presso il *CSC* si e deciso di estrarre e classificare tali discontinuità in maniera automatica, sollevando i tecnici audio e i musicologi da un lavoro ripetitivo, stancante e pieno di errori.

Da qui nasce il lavoro della mia tesi: poter estrarre tramite la tecnica più opportuna tali discontinuità e poterle classificare tramite i più consoni *modelli di reti neurali*. La mia tesi inizia delineando lo stato dell'arte sull'argomento. A seguire saranno affrontate le discusse le problematiche affrontate, le soluzioni sviluppate e i risultati ottenuti.

Capitolo 2

Reti Neurali e problema della classificazione

2.1 Introduzione

Inizialmente, per capire al meglio il motivo delle scelte fatte durante la sperimentazione, è importante fare un piccolo riassunto sullo stato dell'arte della problematica studiata, precedentemente affrontata da due miei colleghi con altre tecnologie. Quest'ultime erano composte da:

- uno script di *frame extraction* capace di individuare ed estrarre *frame* di nastro di video passati come input, raffiguranti potenziali discontinuità, individuate tramite considerazioni fatte sulle variazioni di colore che avvenivano nei pixel durante lo scorrimento del nastro.
- un modello di classificazione (*GoogleNet*) capace di classificare tali discontinuità con buone performance.

Prima di parlare dello script di estrazione, inizierò dando un breve sguardo alle *reti convoluzionali* per evidenziare i punti cardine di tale tecnologia e spiegare per quale motivo esse vengano utilizzate nel campo del *riconoscimento di immagini* e successivamente nella *classificazione*.

2.2 Reti Neurali e Machine Learning

L'apprendimento automatico, noto anche con il termine *machine learning*, è uno dei temi fondamentali dell'intelligenza artificiale. Esso raccoglie un insieme di metodi, sviluppati da più comunità scientifiche: statistica computazionale, il riconoscimento di pattern, il filtraggio adattivo, la teoria dei sistemi dinamici, elaborazione delle immagini, *data mining*, algoritmi adattivi[15].

Questo tipo di tecnologia utilizza metodi statistici per migliorare progressivamente e in maniera iterativa le performance di un algoritmo, al quale viene assegnato il compito di identificare pattern comuni nei dati utilizzati come input.

In questa materia si possono individuare due approcci ben distinti. Il primo metodo porta allo sviluppo di macchine ad apprendimento automatico utili ad un impiego generale, in cui

il comportamento è costituito da una rete connessa casualmente, governata da una routine di apprendimento basata su ricompensa o punizione (*apprendimento per rinforzo*). Questa tecnica punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per raggiungere determinati obiettivi tramite interazione con l'ambiente in cui sono immersi. L'apprendimento per rinforzo si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende soltanto dallo stato attuale del sistema, il quale ne determina quello futuro.

Il secondo metodo, più specifico, consiste nel riprodurre l'equivalente di una rete altamente organizzata progettata per imparare solo alcune attività specifiche. Questa procedura, la quale necessita di supervisione come dice il nome (apprendimento supervisionato), richiede la riprogrammazione per ogni nuova applicazione, ma risulta essere molto più efficiente dal punto di vista computazionale. Esattamente l'opposto si definisce l'*apprendimento non supervisionato*; è una tecnica che consiste nel fornire al sistema informatico una serie di input che riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi (le classi non sono note a priori).

L'apprendimento automatico, attraverso lo studio dei dati, si occupa dello sviluppo di modelli induttivi basati su campioni. Tutto questo è strettamente legato al *machine learning* e viene impiegato in quei campi dell'informatica nei quali, progettare e programmare algoritmi espliciti, è impraticabile.

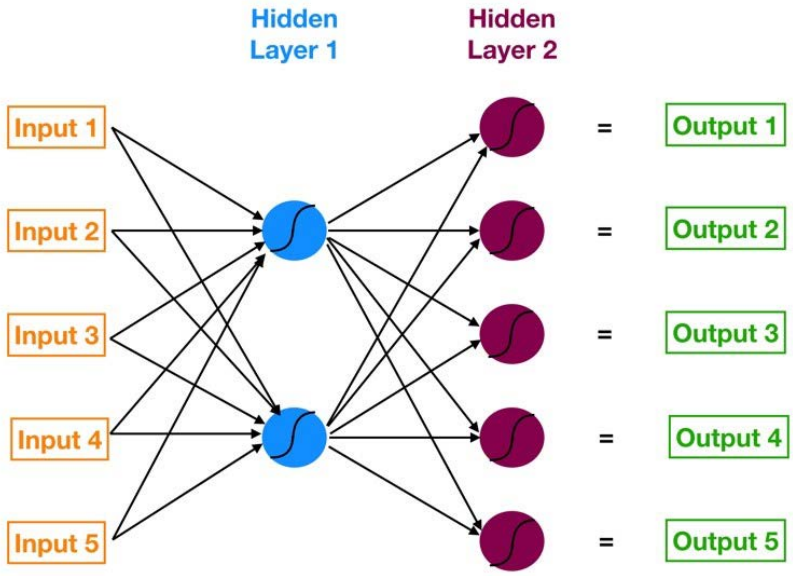


Figura 2.1: Struttura Neural Network [15]

Una rete neurale artificiale (NN "Neural Network" in inglese), è un modello di calcolo matematico/informatico, che prende spunto dalle reti neurali biologiche. Queste *reti neurali* vengono costruite tramite la classica struttura di una generica rete: sono composte da un ***layer* di input**, uno o più ***layer intermedi*** (detti nascosti, o *hidden layer*) e un ***layer* di output**. Ogni uno di questi livelli è costituito da più nodi, i quali possiedono un insieme di ingressi, un' uscita e una funzione

di attivazione (*neurone "acceso" o "spento"*). Questi sono collegati tra loro tramite connessioni pesate, alle quali viene assegnato un valore ***w*** per ogni connessione in input, fornendo una misura in base al loro grado di importanza.

In fase di allenamento la rete apprende, in maniera iterativa, andando a cambiare tali pesi ad ogni singolo ciclo. La tecnica di retropropagazione più utilizzata è quella della discesa del gradiente (*gradient descend*). La rete produce l'output dopo aver espanso l'input attraverso tutti i *layer* della rete. In base a questo compie delle predizioni sui dati, calcolando nei casi più semplici, una ***loss function*** *L(Xi, w)*.

Dopo aver fatto un calcolo di previsione sull'insieme di test, il modello cerca di quantificare quanto l'output prodotto dall'*i-esima* iterazione si discosta da quello desiderato. Una volta calcolato il gradiente, il quale indica la direzione di maggior crescita di una funzione di più variabili, muovendosi in direzione opposta, si può ridurre l'errore descritto prima.

La struttura base può essere riassunta come una rete che riceve segnali esterni su uno strato di nodi d'ingresso (unità di elaborazione, input), ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli (*layer*). Ogni nodo elabora i segnali ricevuti e trasmette il risultato ai nodi successivi (fig. 2.1).

Due dei *layer* più comuni utilizzati per la costruzione di queste reti e che troveremo poi anche nelle CNN sono: livello di attivazione o *ReLU*, e livello di *pooling*. L'unità di rettifica (**ReLU**) consente un addestramento più rapido ed efficace, mappando i pesi negativi a zero e mantenendo soltanto i valori positivi. Talvolta si parla di attivazione, perché solo le *caratteristiche* attivate vengono fatte passare al livello successivo. Inoltre, tale funzione, è molto più efficiente computazionalmente di altre, poichè si tratta esclusivamente di una scelta tra due valori e quindi non necessita di eseguire operazioni particolarmente complesse.

Gli strati di **pooling**, invece, hanno il compito di semplificare l'output eseguendo un sottocampionamento, riducendo il numero di parametri che la rete deve utilizzare per apprendere.

2.3 Reti neurali convoluzionali

Nel *learning* automatico, una CNN o più comunemente chiamata *rete neurale convoluzionale*, è un tipo di rete in cui lo schema di connettività tra i neuroni e i vari *layer* è ispirato all'organizzazione della corteccia visiva animale. Tale tecnologia infatti è spesso utilizzata in materia di classificazione. I singoli filtri sono disposti in maniera tale da rispondere ai vari stimoli derivanti dai livelli precedenti che compongono il *"campo visivo"*. Il loro funzionamento è molto simile a ciò che accade all'interno della corteccia celebrale. L'apprendimento è dettato dai processi biologici che avvengono in maniera tale da ridurre al minimo la pre-elaborazione. Queste teorie hanno diverse applicazioni, soprattutto nel riconoscimento di immagini e video, nella loro classificazione, nei sistemi di raccomandazione, nell'elaborazione del linguaggio naturale e, recentemente, anche in bioinformatica [16].

Come si intuisce dal nome, le CNN adottano la convoluzione come operazione che governa i *layer convoluzionali*. L'operazione nel dominio discreto per due variabili viene definita come segue:

$$(f * g)(x, y) \sum_{u=-\infty}^{\infty} f(u, v)g(x - u, y - v) \tag{2.1}$$

Utilizzando un'architettura *ConvNet*, e addestrando quindi tale rete, si può ottenere una classificazione categorica o numerica. Grazie a queste strutture è possibile estrarre *features (caratteristiche)* anche non facilmente visibili ad occhio umano. E' poi possibile usare tale rete per addestrare un classificatore lineare, il quale successivamente prenderà una decisione basata sul valore di una combinazione lineare di queste caratteristiche. Questi classificatori funzionano bene per problemi pratici, nati in ambienti con molte variabili, raggiungendo livelli di accuratezza paragonabili ai classificatori più complessi (non lineari), richiedendo inoltre meno tempo di *training*.

Si può effettuare l'addestramento di una CNN sia su CPU, su singola GPU, o GPU multiple in parallelo variandone quindi le prestazioni, in particolare i tempi di calcolo.

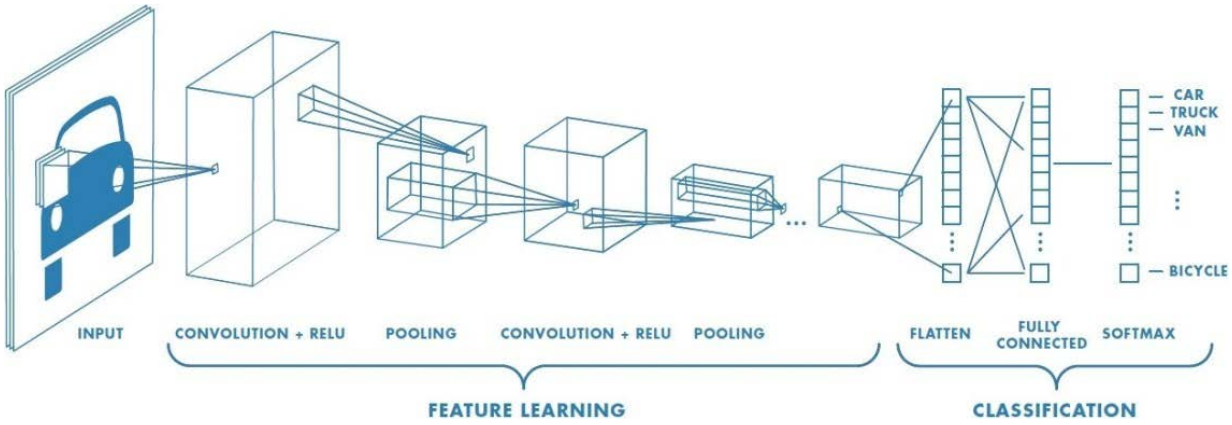


Figura 2.2: Struttura Convolutional Neural Network [16]

2.3.1 Progettazione rete convoluzionale

Come accennato in precedenza, per la classificazione di immagini, è preferibile utilizzare una rete neurale convoluzionale. Questo tipi di modelli, grazie alla loro struttura, ovvero alla concatenazione di più filtri, sono abili nel riconoscere caratteristiche di un'immagine. Questa tipologia di rete deve essere capace di ricevere matrici di pixel come input e dato che i valori di questi pixel sono RGB, si andranno a gestire tre matrici per ogni immagine, una per ogni componente di colore.

Viene spontaneo chiedersi perchè non utilizzare reti *fully connected (completamente connesse)*; in questi casi infatti non ci sarebbe una perdita di informazione. Il problema è che una rete totalmente connessa porta a un'esplosione combinatoria del numero di nodi e di connessioni richieste.

In ogni *livello* è presente una matrice che viene comunemente chiamata ***features map***, ovvero la caratteristica specifica che i nodi di quel determinato *layer* sono incentivati a cercare.

Come in figura 2.3 ad esempio, un primo *layer* potrebbe occuparsi di estrapolare tutte le linee orizzontali dell'immagine. Quindi l'idea principale è quella di far scorrere un apposito filtro sull'immagine di partenza (qui un 2x2), moltiplicandolo poi il tutto per l'area sottostante (prodotto scalare). Questo significa che, in conclusione, all'interno del layer convoluzionale ciascun nodo è mappato solo su un sottoinsieme di nodi di input e, in poche parole, moltiplicare il filtro per il campo recettivo di ciascun nodo equivale concettualmente a sovrapporre tale filtro lungo tutta l'immagine di input (*windowing*).

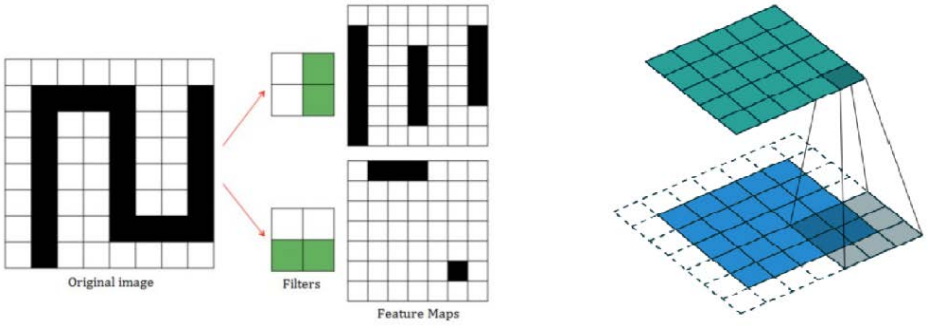


Figura 2.3: Esempio di finestra attuata da un filtro in un layer convoluzionale

Tre dei *layer* più comuni in una CNN sono: *convoluzione*, *attivazione o ReLU*, e *pooling* (già introdotti precedentemente).

La convoluzione applica, alle immagini di input, una serie di filtri convoluzionali, ognuno dei quali attiva determinate *caratteristiche*. Come detto in precedenza, i vari *layer* di neuroni che si occupano di questo, dividono tutta l'immagine in vari frammenti sovrapposti, i quali individueranno dei pattern caratterizzanti trasferendo l'informazione allo strato seguente sotto forma di una *feature map*. Ad ogni iterazione sul training, i valori presenti all'interno di ogni una di queste matrici cambia, poichè la rete ciclo dopo ciclo impara a riconoscere e ad estrarre caratteristiche più efficaci. Ogni livello di convoluzione aumenta la complessità nel modello e, di conseguenza, il volume in output. La dimensionalità quindi diventa sempre più intrattabile aumentando la profondità della rete. Inoltre, in una rete complessa, può presentarsi anche la problematica dell'*overfitting*. Essa scaturisce quando il modello inizia ad imparare *caratteristiche* che sono troppo speciche al relativo training set di partenza, e che magari non hanno riscontro nella totalità dei casi.

Nel caso di training su immagini, questo processo è molto simile ad una riduzione dei pixel totali: ricapitolando l'immagine viene divisa in regioni (con sovrapposizioni) dette finestre, le quali vengono rappresentate da un unico pixel. Ad esse viene assegnato un colore in base alla media dei valori dei canali RGB della regione di partenza (*average pooling*) o, in alternativa, si può scegliere di assegnare al nuovo pixel il valore massimo da ogni regione (*max pooling*). Dopo aver appreso e definito le *features maps* dei vari *layer*, l'architettura di una CNN passa alla fase di classificazione.

Il penultimo *layer*, molto spesso, è un livello completamente connesso che genera un vettore di K elementi, la cui dimensione è il numero di classi che la rete sarà in grado di distinguere

e classificare. Le componenti di questo vettore ne identificano le probabilità di appartenenza. L'architettura CNN utilizza un *layer* di classificazione come softmax per fornire l'output del problema.

2.4 Tecnologie Utilizzate

Al momento di scegliere le tecnologie da utilizzare per raggiungere l'obiettivo di questo progetto, sono state considerate due possibili alternative: configurare l'elaboratore nella sua totalità e quindi lavorarci direttamente per ogni parte del progetto, creando un macro ambiente nel quale saremo stati capaci di sviscerare ogni tematica di studio, oppure utilizzare piccoli *environment* dedicati per ogni sottoprogetto. All'interno di questi, si sarebbero utilizzate soltanto le librerie opportune cercando di ottimizzare al meglio le prestazioni della macchina. Il lavoro è stato svolto seguendo la seconda strada creando micro-Environment utili a tale scopo, in modo tale da facilitare il *debugging* riguardante l'incompatibilità di versione dei pacchetti e librerie di programmazione. Una delle piattaforme maggiormente utilizzate per la creazione di questi ambienti di sviluppo è **Anaconda Enterprise**.

2.4.1 Anaconda Enterprise

Immaginiamo di dover condividere tra diversi *data-scientist* regolarmente progetti di AI e *machine learning*, soprattutto per la produzione su larga scala, fornendo rapidamente informazioni dettagliate su di essi. *Anaconda Enterprise* è una piattaforma capace di fare tutto questo, in grado di scalare facilmente da un computer di un singolo utente, a migliaia di macchine connesse tra loro. Anaconda fornisce gli strumenti necessari per:

- Raccogliere dati da *file*, *database e data-lake*.
- Gestire ambienti con Conda.
- Condividere, collaborare e riprodurre progetti .

Anaconda è una distribuzione gratuita e *open source* dei linguaggi di programmazione Python e R per il calcolo scientifico, che mira a semplificare la gestione dei pacchetti attraverso ambienti di sviluppo distribuibili poi semplicemente attraverso file *yaml*. Le versioni dei pacchetti e delle librerie sono gestite dal sistema di gestione del pacchetto *conda*. La distribuzione Anaconda include pacchetti di *data science* adatti per tutte le piattaforme (Windows, Linux e MacOS). Questo piattaforma rende possibile e semplice l'utilizzo di **Jupyter Notebook**; questa è un'applicazione Web *open source* che consente di creare e condividere in tempo reale script e modelli di programmazione. Gli usi includono: pulizia e trasformazione dei dati, simulazione numerica, modellistica statistica, visualizzazione dei dati, apprendimento automatico e molto altro.

2.4.2 Python e Tensorflow

Come accennato in precedenza, una volta creato l'ambiente *"contenitore"* di sviluppo, bisogna decidere quali pacchetti installarvi al suo interno, in base ovviamente all'obbiettivo da raggiungere. Nei paragrafi sottostanti verranno citate tutte le versioni utilizzate di tali librerie, con lo scopo di aver una facile riproducibilità delle varie sperimentazioni svolte. Conda rende disponibile anche l'utilizzo del pacchetto Python per la programmazione (versione 3.6.4). Ad esso viene agganciata Tensorflow (1.14), una libreria software *open source* per l'apprendimento automatico, la quale fornisce strutture utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi. Questa libreria può funzionare su numerosi tipi di CPU e meglio ancora su GPU, grazie al supporto di linguaggi come CUDA. TensorFlow ha due caratteristiche fondamentali:

- **Facile costruzione dei modelli:** la costruzione e l'addestramento di modelli *machine learnig* è facile ed intuitiva. Usando API di alto livello come Keras, con un'esecuzione sistematica e facile da assimilare, l'interazione tra utente e modello è veloce consentendo un *debug* rapido ed intelligente.

- **Produzioni di modelli *machine learning* in qualsiasi ambiente:** è possibile costruire e distribuire facilmente i diversi modelli tra i vari *cloud*, tra i *browser* o su diversi dispositivi, indipendentemente dal linguaggio utilizzato.

2.5 Rete GoogleNet

GoogleNet [23] è la prima rete convoluzionale che è stata studiata e analizzata per il riconoscimento delle discontinuità. E' stata vincitrice della competizione ILSVRC 2014 (*ImageNet Large Scale Visual Recognition Challenge*), risultando la più efficiente tra quelle presentate, ottenendo come accuracy top-1 68.7% e accuracy top-5 88.9%. La rete e costituita da un centinaio di *layer* ed è stata allenata e testata su un dataset suddiviso in 1.2 milioni di immagini per il *training set*, 50.000 per il *validation set* e 100.000 per il *test set*. Essa è stata allenata con immagini di dimensione (224, 224, 3) per un totale di 1000 classi in output [23].

I primi strati di rete che l'immagine in input deve attraversare sono una serie di livelli convoluzionali, *pooling* e di normalizzazione (come le classiche reti neurali convoluzionali). Questo primo blocco viene definito ***stem***, al quale segue poi il blocco cardine della struttura della GoogleNet, il modulo ***inception***(figura 2.6). Anch'esso è composto da una serie di *layer* convoluzionali e di *pooling*, posti in parallelo e poi concatenati tra loro. La rete contiene nove di questi moduli, inframmezzati da due *layer* di *max pooling*. Come accennato precedentemente, questi servono a ridurre le dimensioni delle *matrici di features* che di volta in volta attraversano la rete [4].

All'interno del modulo *inception* vengono utilizzate delle convoluzioni 1x1. Esse servono a ridurre la dimensionalità degli input per i livelli posti successivamente, i quali presentano dimensioni di filtro maggiori (5x5 o 3x3). Questo approccio si traduce in un modello ad alte prestazioni con un numero minore di parametri rispetto ad altre *convolutional neural network*.

A causa della profondità della rete, sono stati inseriti due output ausiliari, uno circa a metà del modello e l'altro a due terzi. Lo scopo di questi classificatori è di attenuare il problema della

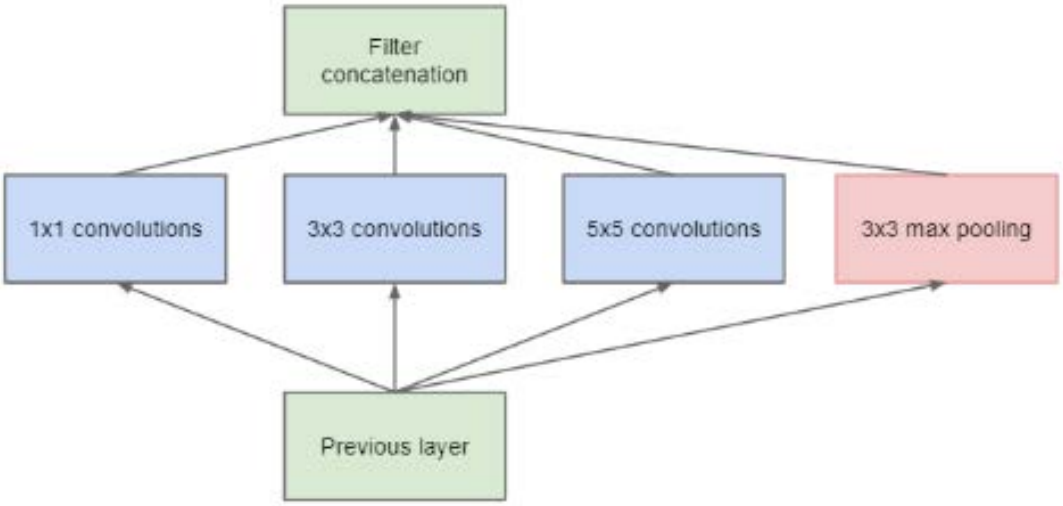


Figura 2.4: Modulo inception[24]

scomparsa del gradiente, controllandolo durante l'attraversamento. Infine, si trova il modulo di output, formato da un *layer di average pooling* seguito da una funzione di attivazione *softmax* su un *fully connected layer*.

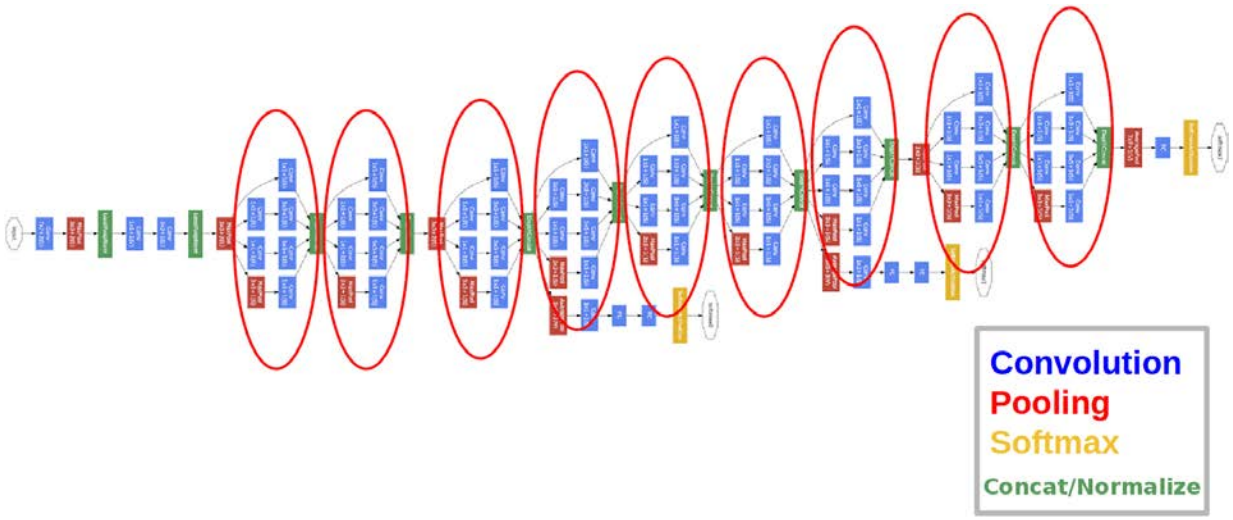


Figura 2.5: Struttura GoogleNet

2.5.1 Integrazione della Keras API specification

Durante gli anni, il progetto di sviluppo di TensorFlow 2.0, è stato guidato dai vari feedback arrivati dalla community di utenti. Ecco perché si è deciso di inserire all'interno di questa *build* l'integrazione completa con le *Keras API specification* in modo tale di avere un'interfaccia sovrastante con la quale interagire più facilmente.

Keras è stata pensata per la **creazione di prototipi veloci** e dispone di tre caratteristiche chiave:

- **User-friendly:** fornisce delle chiare e semplici funzioni da richiamare per costruire e compilare il modello.
- **Modulare:** i *Keras model* sono stati concepiti per essere connessi tra di loro senza troppe restrizioni.
- **Facile da estendere:** Keras permette di scrivere nuovi blocchi di lavoro personalizzati, *layer*, metriche, *loss function* e di sviluppare modelli in modo molto semplice.

Keras contiene già al suo interno numerose implementazioni di blocchi di rete neurale comunemente usati come *layer*, funzioni di attivazione, ottimizzatori e una serie di strumenti per semplificare il lavoro normalmente svolto con un dataset di immagini. Grazie a questa API siamo in grado di semplificare la codifica necessaria utile a costruire e strutturare una *Deep Neural Network* su smartphone (iOS e Android), su Web o su Java Virtual Machine. Il codice è presente su GitHub [24].

Oltre alle reti neurali standard, Keras supporta quelle convoluzionali e ricorrenti, rende facile l'implementazione di altri livelli di utilità comune come *layer* di *dropout, batch normalization e pooling.*

2.6 OpenCv

Dato che l'obbiettivo della mia tesi è estrarre tramite la tecnica più opportuna discontinuità presenti nei video analizzati e poterle poi classificare tramite i più consoni modelli di reti neurali, è necessario fare una discussione sulla costruzione e la composizione del **Dataset** da usare. Durante l'estrazione dei frame è stata usata la libreria *OpenCV* (Open source Computer Vision library). Questa è una libreria software multipiattaforma (Windows, Mac OS, Unix, Android) nell'ambito della *computer vision*. Il linguaggio nativo di tale libreria è il C++, ma esistono estensioni per Python, Java e Mathlab. OpenCV è accessibile e modificabile da tutti grazie alla licenza BSD (*Berkley Software Distribution*). Chi apporta variazioni a un programma protetto da questa licenza, può ridistribuirlo senza l'obbligo di indicare tutte le modifiche apportate al codice sorgente.[25]

Sono circa 500 le funzioni messe a disposizione da OpenCV, e sono tutte finalizzate all'elaborazione di immagini, al t*racking* e all'o*bject detection*. L'immagine può venire rappresentata in tre maniere diverse:

- in bianco e nero;

- in scala di grigi;
- a colori, in quest'ultimo caso OpenCV di default utilizza lo spazio di colori BGR, una permutazione dei tre classici canali RGB.

2.7 Ottimizzatore Gradient Descent

Il concetto di ottimizzazione gioca un ruolo chiave quando si parla di machine learning e deep learning in particolare. Lo scopo principale degli algoritmi di deep learning è quello di costruire un modello di ottimizzazione che, tramite un processo iterativo, minimizzi o massimizzi una funzione obiettivo denominata anche *loss function* o *cost function*; in questo paragrafo parleremo dell' *ottimizzatore GD* o meglio conosciuto con il termine *gradient descent*. Con il termine gradiente, in poche parole, si intende la *pendenza o inclinazione*. Quindi, quando si parla di **discesa del gradiente**, significa letteralmente scendere da tale pendenza per raggiungere il punto più basso. E' comune usare il problema della regressione lineare per spiegare quest'algoritmo. L'obiettivo della regressione è di minimizzare la somma degli scarti quadratici medi. Sapendo che una funzione raggiunge il suo valore minimo quando la pendenza è uguale a 0, usando questa tecnica riusciamo trovare il vettore tale per cui la funzione raggiunge tale valore. Lo stesso risultato può essere trovato con la tecnica della discesa del gradiente. I passaggi dell'algoritmo sono i seguenti:

- Trovare la pendenza della funzione obiettivo rispetto a ciascun parametro. In altre parole si calcola il gradiente della funzione.
- Scegliere un valore iniziale casuale per i parametri.
- Aggiornare la funzione gradiente inserendo i valori dei parametri.
- Calcolare le dimensioni del passo per ogni funzione come:

$$StepSize = Gradient * LearningRate$$

- Calcolare i nuovi parametri come:

$$NewParams = OldParams - StepSize$$

- Ripetere i passaggi da 3 a 5 fino a quando non si è raggiunto l'ottimo.

Il ***learning rate*** sopra menzionato è un parametro flessibile che influenza fortemente la convergenza dell'algoritmo. *Learning rate* più elevati fanno sì che l'algoritmo si avvicini all'ottimo troppo velocemente e quindi potrebbe saltare oltre il punto minimo mancando tale soluzione.

Quindi è sempre efficace attenersi a un basso tasso di apprendimento come 0,001 nelle prime fasi del *training*. Si può anche dimostrare matematicamente che l'algoritmo di *Gradient Descend* fa passi più "lunghi" lungo la pendenza se il punto di partenza è distante dal minimo, e passi più corti quando si avvicina alla destinazione, quindi il rischio di perderlo o passarci oltre è enorme.

2.8 Training

Durante l'allenamento di una rete ci si trova a configurarne alcuni parametri, e grazie alla loro variazione e controllo si possono determinarne le performance di un modello. Alcuni di questi sono i seguenti:

- Un'**iterazione** è composta da un *forward pass* e/o un *backward pass* di ogni campione preso in esame. Nel *forward pass*, viene calcolata la **loss function** mano a mano che le matrici di *features* che attraversano tutti i livelli della rete. Nella *backward pass*, invece, la rete aggiusta i pesi trovati nella fase precedente e di fatto, impara.

- Un'**epoca** identifica il numero di volte nel quale l'algoritmo osserva l'intero *training set*. E' composta anch'essa da un *forward pass* e un *backward pass* (di tutti i dati di *training*).

- La **Batch size** è il numero di *training sample* che l'algoritmo osserva in un'iterazione. Più alto si inizializza tale valore, più spazio in memoria è necessario. Molto spesso si decide di non osservare l'intero dataset proprio per ridurre le dimensioni della rete e occupare quindi meno RAM durante l'elaborazione dei dati, in base ovviamente alle risorse disponibili. In un'epoca, la totalità del Dataset deve sempre essere osservata e quindi se si hanno 1000 campioni di training e si imposta un batch size pari a 500, saranno necessarie due iterazioni per completarne una.

- Dato che l'obiettivo ultimo della rete neurale è quello di minimizzare la *loss function*, matematicamente possiamo definire la funzione di costo come *L(Xi, w)*, dove *Xi* indica l'i-esima iterazione e *w* i pesi che la rete apprende. I pesi vengono poi aggiornati secondo la regola:

  $$w := w - \eta \nabla L$$

  η è chiamato ***learning rate*** (tasso di apprendimento) e determina quanto velocemente la rete aggiorna i parametri durante il *training*. Solitamente si imposta un *learning rate* iniziale (circa 0.001) e lo si aggiorna dopo un certo numero di iterazioni, moltiplicandolo per una costante minore di 1.

- Gli **ottimizzatori** cercano di adattare il modello aggiornandolo a seconda dell'output seguendo e considerando la funzione di costo. Uno degli ottimizzatori piu usati è la *discesa del gradiente* (*Gradient Descent - GD*), ed è quella che useremo nei nostri primi esperimenti. Comunque a volte è più ragionevole utilizzare solo un sottoinsieme dei campioni di training, il batch, su cui calcolarne il gradiente. Il nome a cui viene data questa tecnica è *discesa stocastica del gradiente* (*StochasticGradient Descent- SGD*).

- La **regolarizzazione** è una fase presente nel processo di ottimizzazione svolta con lo scopo di evitare l'*overfitting*. Questa pratica inoltre aggiunge una penalizzazione alla funzione di costo per i pesi con valore troppo elevato. Questo porta una generalizzazione migliore sui nuovi dati. Un esempio di questa pratica di regolarizzazione è il *DropOut*, il quale consiste nell'impostare una frazione di unita di input a 0 ad ogni aggiornamento durante l'allenamento.

2.9 Transfer learning

Il *transfer learning* è una tecnica basata sull'apprendimento automatico che si focalizza nella memorizzazione di relazioni tra input e output, pattern e modelli, acquisite durante la fase di training su un problema non correlato con quello preso in esame, per poi metterle in relazione con quest'ultimo [9].

L'obiettivo è quello di riutilizzare o trasferire informazioni da ***learned tasks***, precedentemente appresi, per l'apprendimento di nuovi. Tutto questo ha portato un significativo incremento del numero e della tipologia delle reti utilizzate nell'ambito del *machine learning*. Questa tecnica inoltre cerca di migliorare l'efficienza del singolo campione del dataset.

"Transfer learning and domain adaptation refer to the situation where what has been learned in one setting is exploited to improve generalization in another setting"

È possibile utilizzare questa tecnica di apprendimento su problemi molto spesso di natura predittiva. Sono due gli approcci comuni più frequenti [11]:

- **Develop Model Approach:** Questo approccio si divide in più fasi. Inizialmente è necessario selezionare un problema di modellazione predittiva con un'abbondanza di dati nella quale possa esistere una relazione tra dati di input e dati di output. Successivamente è fondamentale sviluppare un modello abile per risolvere tale problema. Quest'ultimo deve essere migliore di un modello *naive* costruito sul problema di partenza, per garantire che sia stato eseguito un vero e proprio apprendimento delle funzionalità. Il modello così creato ed allenato può quindi essere poi utilizzato come punto di partenza per la seconda attività di interesse. Ciò può comportarne l'utilizzo totale o parziale, a seconda della tecnica di modellizzazione utilizzata. Facoltativamente poi potrebbe essere necessario adattare o affinare il modello sui dati della coppia input-output disponibili per il *task* di interesse.

- **Pre-trained Model Approach:** Anche questo approccio si divide in più fasi. Un modello sorgente pre-addestrato viene scelto tra quelli disponibili. Molti istituti di ricerca ne rilasciano di già pre-allenati su *set* di dati di grandi dimensioni, i quali possono essere inseriti nel *pool* di modelli candidati tra cui scegliere. Uno di questi può quindi essere utilizzato come punto di partenza nell'attività di interesse. Ciò può comportare anche qui l'utilizzo totale o parziale del modello, a seconda della tecnica di modellazione utilizzata. Questo secondo tipo di apprendimento per "trasferimento" è comune nel campo del *deep learning*.

In generale, non è ovvio che ci sarà un vantaggio nell'utilizzare questo approccio fino a quando il modello non sarà sviluppato e valutato. Il *Transfer Learning* è un'ottimizzazione, una scorciatoia per risparmiare tempo e in caso ottenere prestazioni migliori.

E' una tecnica da provare solo in alcuni casi; se è possibile identificare nel dataset a disposizione una qualche correlazione con il dataset ampio di partenza su cui il modello è già stato preallenato, oppure se è disponibile già un modello pre-addestrato che è possibile utilizzare come punto di partenza. Ed è proprio quest'ultimo il caso in cui noi ci troviamo. Lisa Torrey e Jude Shavlik, nel loro capitolo sul *transfer learning* descrivono tre possibili vantaggi da identificare quando si utilizza questo approccio [11]:

- **Higher start:** Le performance iniziali sul modello di origine (prima di affinarlo) sono superiori a quelle che sarebbero altrimenti.
- **Higher slope:** Il tasso di miglioramento durante l'allenamento del modello sorgente è più ripido di quanto non sarebbe altrimenti.
- **Higher asymptote:** L'abilità convergente del modello addestrato è migliore di quanto sarebbe altrimenti.

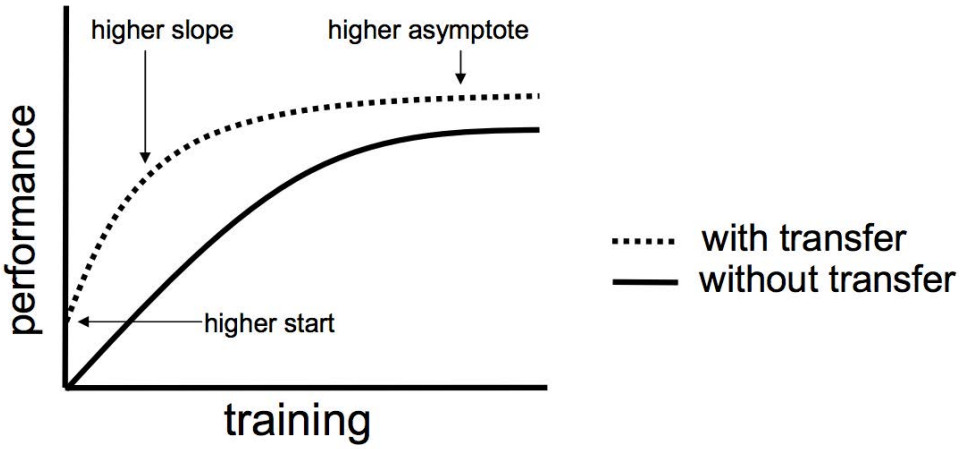


Figura 2.6: Tre casi in cui Transfer Learning può migliorare l'apprendimento [11]

2.10 Keras e modelli pre-allenati

Le *Keras application* sono modelli pre-allenati di *Deep Learning* resi disponibili insieme ai loro pesi (*Pre-trained Model Approach*). Questi modelli possono essere utilizzati per la classificazione, il *feature extraction*, e il *fine-tuning*. Nella tabella 3.1 sono riportati brevemente quelli utilizzati in questo elaborato con le rispettive dimensioni in termini di parametri interni e allenabili. Nei paragrafi successivi spiegherò brevemente le differenze sostanziali tra le varie reti prese in considerazione.

Tabella 2.1: Modelli e numero di parametri che li compongono.

Model	Parameters
MobileNet [5]	4,253,864
MobileNetV2 [10]	3,538,984
DenseNet121 [6]	8,062,504
EfficientNet [14]	4,053,407
SqueezeNet [13]	724,035

2.10.1 MobileNet e MobileNetV2

Questi due modelli fanno parte di una classe di *reti neurali* efficienti, utilizzate frequentemente per applicazioni di ***mobile vision*** ed ***embedded***. Dato che, come spiegato in precedenza, l'aspetto negativo delle reti convoluzionali è l'eccessiva dimensionalità derivante dall'enorme numero di connessioni tra i neuroni dei vari livelli, esse vengono strutturate seguendo un'architettura ottimizzata che utilizza convoluzioni separabili, le quali spiegheremo successivamente, in modo tale da costruire reti neurali sempre più leggere. Introducendo inoltre due iperparametri globali poco complessi si dà la possibilità alla *rete* di adattarsi in base alla problematica analizzata e dai vincoli trovati (moltiplicatore di profondità e fattore di espansione). E' stata dimostrata quindi l'efficacia delle MobileNets in una vasta gamma di applicazioni tra cui: rilevamento di oggetti, classificazione, *pattern matching* di volti e geolocalizzazione. MobileNet e MobileNetV2 sono entrambe presenti nel modulo di applicazioni di Keras. Quest'ultima offre una classificazione immediata delle immagini, per entrambe le reti, quando la categoria che si desidera prevedere è disponibile già nelle classi di ImageNet. Altrimenti, come spiegato in precedenza, grazie al *Fine-Tuning*, si può ottimizzare le rete per il set di dati e le classi di cui siamo a disposizione.

L'idea alla base di MobileNetV1 (versione 1) è di definire dei blocchi convoluzionali, essenziali per le attività di *computer vision* (sono piuttosto costosi da calcolare), che rappresentino le cosiddette convoluzioni separabili in profondità [5]. Innanzitutto possiamo notare che il lavoro svolto da questo blocco può essere diviso in due sotto-attività: la prima, chiamata *Depthwise Convolution*, filtra l'input, mentre la seconda, costituita da un livello di convoluzione 1x1 (*Pointwise Convolution*), combina questi valori per creare nuove *features* utili per l'apprendimento.

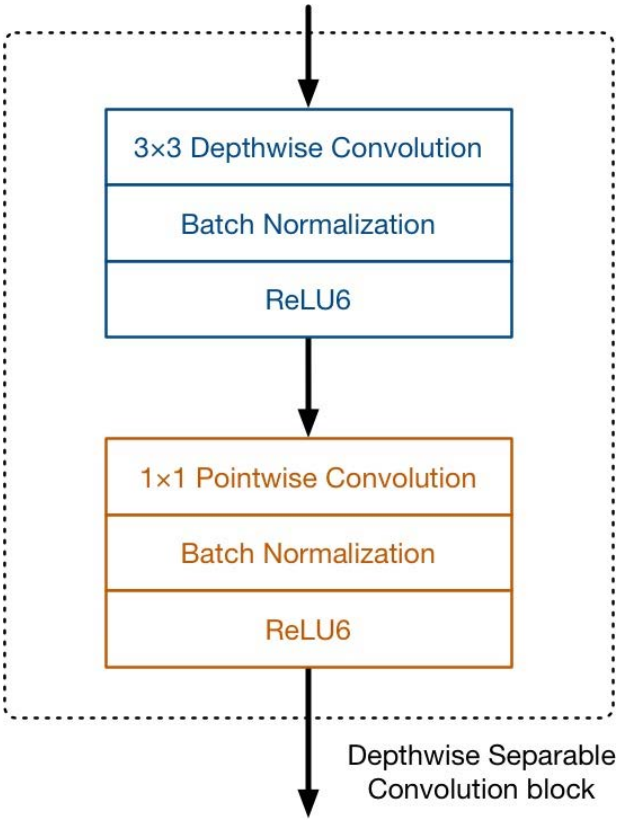


Figura 2.7: Convoluzione separabile in profondità V1 [5]

Insieme, i due livelli formano un blocco di convoluzione *Depthwise Separable Convolution*. Il risultato ottenuto ha le stesse funzionalità della convoluzione tradizionale ma molto più veloce e facile da calcolare.

L'architettura completa di MobileNetV1 è costituita da una normale convoluzione 3x3 come primo strato, seguita da 13 ripetizioni del blocco di cui abbiamo parlato sopra. Non ci sono strati di *pool* tra questi blocchi piuttosto, in alcuni dei livelli più profondi, viene definito un passo (*stride*) di 2 per ridurre le dimensioni spaziali dei dati. Quando ciò accade, il livello puntuale corrispondente raddoppia anche il numero di canali di uscita. Anche qui, come è comune nelle architetture moderne, gli strati di convoluzione sono seguiti sempre da *Batch di normalizzazione*.

In un classificatore basato su MobileNet, in genere, esiste sempre un livello di *Global Average Pooling* al termine del blocco di calcolo, seguito poi da un livello completamente connesso (o da una convoluzione 1x1 equivalente) e da un *softmax*. Come detto in precedenza esistono diversi iperparametri che consentono di definire la dimensionalità dell'architettura. Il più importante di questi è il **moltiplicatore di profondità** il quale modifica il numero di canali in ciascun livello. L'uso di un moltiplicatore di profondità pari a 0,5 ad esempio dimezza il numero di canali utilizzati dai *layer*, riducendo di conseguenza il numero di calcoli totali per l'apprendimento. Il modello risultante quindi è molto più veloce di quello completo di partenza ma anche meno preciso [10]. Grazie all'introduzione delle *Depthwise Separable Convolution*, MobileNet impie-

gherà circa 9 volte meno tempo per completare il training rispetto a reti neurali comparabili con la stessa precisione.

MobileNetV2 utilizza ancora la *Depthwise Separable Convolution*, ma il blocco predefinito ora è come quello rappresentato in figura 3.3.

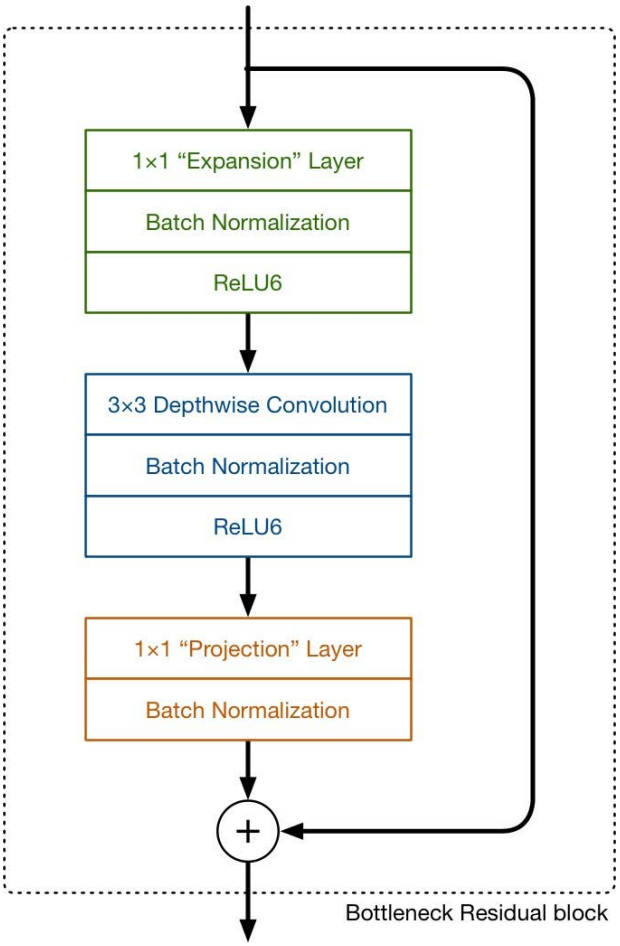


Figura 2.8: Convoluzione separabile in profondità V2 [10]

Come si può notare, in questo caso sono presenti tre livelli convoluzionali. Gli ultimi due sono quelli che già conosciamo: una convoluzione in profondità che filtra gli input, seguita da un livello di convoluzione 1x1. Questo *livello* è noto come *projection layer*. Esso proietta i dati con un numero elevato di dimensioni (canali), in un tensore con un numero di componenti ridotto. Questo tipo di livello va a ridurre di volta in volta la quantità di dati che fluisce attraverso la rete.

Il primo strato, come in MobileNetV1, è una convoluzione 1x1. Il suo scopo è quello di espandere il numero di canali ***(expansion layer)*** prima che entrino nella *Depthwise Convolution*. Il **fattore di espansione** è un altro di quei iperparametri usati nella sperimentazione per variare l’architettura in base alle necessità data dal problema di partenza. Il fattore di espansione predefinito è 6: ad esempio, se c’è un tensore con 24 canali che vanno in un blocco, il livello di espansione lo converte prima in un nuovo tensore con 24 * 6 = 144 canali. Successivamen-

te, il blocco di *Depthwise Convolution* applica i suoi filtri a quel tensore di 144 canali, il quale verrà poi filtrato dal livello di proiezione su un vettore più piccolo. Quindi ricapitolando l'ingresso e l'uscita dell'intero blocco sono tensori a bassa dimensione, mentre la fase di filtraggio che avviene all'interno del blocco, viene eseguita su un tensore ad alta dimensione. Come in MobileNetV1, ogni *layer* di questa seconda versione di rete, ha a seguire un livello di *batch normalization* e una funzione di attivazione *ReLU6*. Tuttavia, all'output del livello di proiezione, non viene applicata nessuna funzione di attivazione poiché, questo strato, produce dati a bassa dimensione. Diversi esperimenti hanno infatti dimostrato che, facendo l'opposto, si andava a distruggere informazioni utili per l'apprendimento [10]. L'uso di tensori di bassa dimensione è la chiave per ridurre il numero di calcoli. Dopotutto, più piccolo è il tensore, minori sono le moltiplicazioni che gli strati convoluzionali dovranno svolgere. L'applicazione di uno strato convoluzionale, con lo scopo di filtrare un vettore a bassa dimensione non sarà in grado di estrarre molte informazioni utili al *training* del modello. Quindi è necessario idealmente lavorare inizialmente con tensori con un grande numero di componenti per poi ridurre la loro dimensionalità. Il *design* a blocchi di MobileNetV2 ci offre il meglio usando entrambe le idee.

2.10.2 DenseNet

Allo stato dell'arte [6], diversi lavori hanno dimostrato che alcune reti convoluzionali, concatenando tra loro più *dense layer*, possono diventare rapidamente molto profonde. Il motivo per il quale si decide di utilizzare tali livelli è perchè, con le varie sperimentazioni, si è notato che reti con tale struttura sono più facili da addestrare (estrapolazione di *features* più efficace) e più accurate; esse contengono infatti connessioni più brevi tra i *layer* vicini all'ingresso e quelli vicini all'uscita. Di seguito analizziamo questa osservazione e introduciamo la rete CNN densa (DenseNet [6]).

Essa ha una struttura molto semplice, collega ogni livello a ogni altro *layer* in modo **feed-forward**. Per ogni *layer*, le *feature maps* di tutti i livelli precedenti vengono utilizzate come input per i quelli successivi. Mentre le reti convoluzionali tradizionali con *L* strati hanno *L* connessioni (una tra ogni strato e il suo livello successivo), questa nuova rete ha *L*(L + 1)/2* connessioni dirette.

DenseNet presenta numerosi vantaggi:

- Attenua il problema del **vanishing-gradient**; questo problema è un fenomeno che crea difficoltà nell'addestramento delle reti neurali (*deep*). In tutti quei casi in cui abbiamo una **discesa stocastica del gradiente**, ogni parametro del modello riceve un decremento proporzionale alla derivata parziale della *loss function* rispetto al parametro stesso. Poiché nell'algoritmo di retropropagazione i gradienti ai vari livelli vengono moltiplicati tramite la regola della catena, il prodotto di *n* numeri compresi nell'intervallo [0,1] decresce esponenzialmente rispetto alla profondità *n* della rete.

- rende la propagazione delle caratteristiche attraverso la rete più efficace.

- incoraggia il riutilizzo di *features* e riduce sostanzialmente il numero di parametri.

L'effetto controintuitivo di questo modello denso di connettività è che richiede meno parametri da allenare rispetto alle reti convoluzionali tradizionali. Non è infatti necessario riapprendere mappe di funzionalità ridondanti. Anche l'architettura ResNets rendeva vera e tangibile questa metodologia di propagazione ma le varie sperimentazioni mostravano che molti di questi *layer* contribuivano molto poco o venivano ripetuti tra i vari livelli. Questi infatti potevano essere casualmente eliminati durante l'allenamento.[26]

Per questo, con l'architettura DenseNet proposta, si è cercato di distinguere esplicitamente le informazioni **aggiunte** alla rete e quelle **conservate**. I livelli di DenseNet, essendo molto stretti tra loro, aggiungono solo un piccolo set di *feature*, alla "conoscenza collettiva" della rete, alla volta e mantengono invariate le *feature-maps* rimanenti. Sarà poi il classificatore finale a prendere una decisione basandosi su tutte tabelle rimaste nella rete.

Oltre a migliorare l'efficienza dei parametri, un grande vantaggio di DenseNet è il miglioramento del flusso di informazioni e gradienti che si propagano in tutta la rete durante l'apprendimento. Ogni *layer* ha un accesso diretto ai gradienti dalla *loss function* e al segnale di ingresso originale, avendo così a disposizione una ***supervisione profonda implicita*** (non c'è solo uno scambio di *features* tra *layer* adiacenti, *layer* iniziali possono avere un collegamento con i livelli finali della rete). Inoltre, un ulteriore pregio per queste reti, è dato dal fatto che al loro interno possiamo osservare anche connessioni dense che hanno principalmente solo un effetto regolarizzante, il quale riduce il sovra-adattamento ai dati dati come input, quando si hanno dimensioni di *training set* molto piccole (riduzione *overfitting*).

2.10.3 EfficientNet

Questa rete non si concentra solo sul miglioramento della precisione, ma anche sull'efficienza dei modelli. Molto spesso il ridimensionamento della rete porta ad un peggioramento delle performance. Sebbene a volte i ricercatori non si preoccupino molto della dimensionalità del modello ma più sull'obiettivo di migliorare lo stato dell'arte, alcune volte il ridimensionamento, se fatto correttamente, può anche aiutare a migliorare le prestazione di una rete [14]. Come si poteva intuire dai paragrafi precedenti, esistono tre parametri inerenti al ridimensionamento di una CNN:

- **profondità**, sta ad indicare semplicemente quanto è profonda la rete ed equivale al numero di strati in essa contenuti.
- **larghezza**, sta a rappresentare quanto è ampia la rete. Una sua misura ad esempio può essere il numero di canali in un livello Conv.
- **risoluzione**, che indica il dettaglio dell'immagine che viene passata.

Il ridimensionamento del modello, basandosi su una variazione della profondità di questo, è il metodo più comunemente utilizzato per intervenire e risolvere tale problematica. Queste scelte posso derivare da intuizioni che sono diverse per ogni problema affrontato; una rete più profonda ad esempio può acquisire funzionalità (*features*) più ricche e complesse e generalizzare in maniera sempre più ottimale su nuove attività e campioni del *test set*. Teoricamente, con

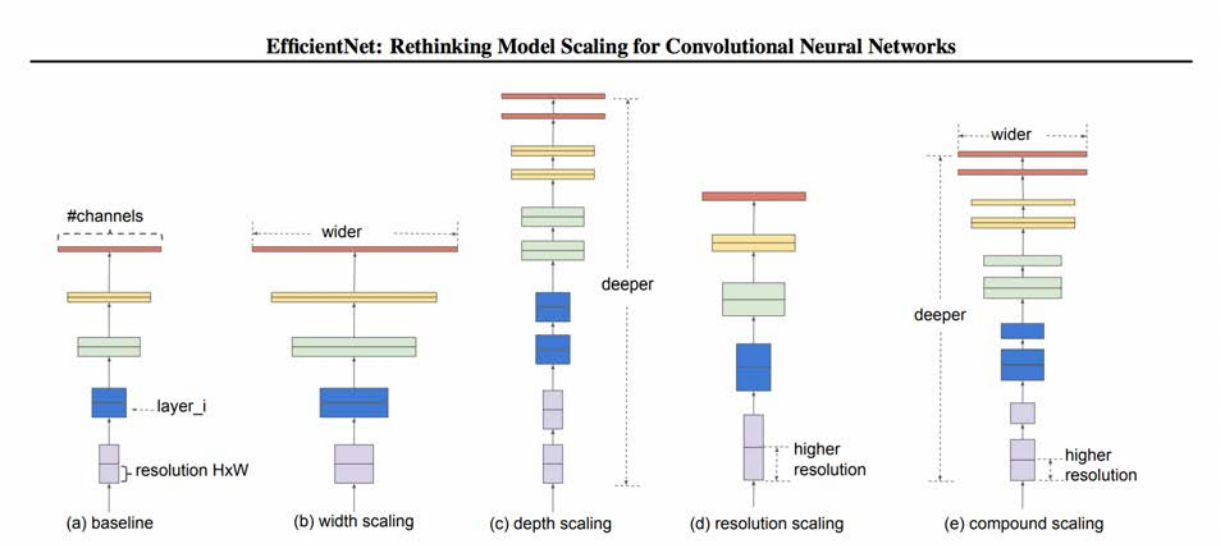


Figura 2.9: Struttura EfficientNet [14]

più livelli, le prestazioni della rete dovrebbero migliorare, ma praticamente questo non avviene. Come spiegato in precedenza, il problema del *vanishing-gradient* è una delle motivazioni più diffuse e comuni che sorgono quando rendiamo la rete troppo profonda.

Il ridimensionamento dei pesi (larghezza) di una rete è comunemente usato quando si decide di mantenere il modello il meno esteso possibile. Anche se reti più ampie tendono ad acquisire funzionalità sempre più dettagliate ed efficaci, i modelli più piccoli sono più facili da addestrare in termini di tempo e spazio. Tutto sommato è l'obiettivo che lo stato dell'arte cerca di raggiungere; modello piccolo e maggiore precisione. Se però tendiamo ad allargare troppo il modello, finiremo per rendere la rete estremamente ampia, con modelli poco profondi, rendendo la rete molto pesante ma con un miglioramento marginale delle prestazioni. Quindi abbiamo capito che non possiamo ne aumentare troppo la larghezza della rete, ne diminuirla in maniera eccessiva. Quindi l'idea che va per la maggiore è quella di collegare e mettere in relazione i ridimensionamenti di *larghezza* ed *ampiezza* così da poter cercare un punto di equilibrio.

Ci interessa però anche dare un'occhiata all'ultima tipologia di ridimensionamento ovvero la *risoluzione*. Possiamo dire che in un'immagine ad alta risoluzione, le caratteristiche sono più dettagliate e quindi a lungo andare queste dovrebbero funzionare meglio nel training a livello di performance ed è uno dei motivi per cui in attività complesse, come il rilevamento degli oggetti, utilizziamo risoluzioni di immagine come 300x300, 512x512 o 600x600. Questo però non è un miglioramento che avviene in maniera lineare; il guadagno di precisione diminuisce molto rapidamente, e ad esempio, l'aumento della risoluzione da 500x500 a 560x560 non comporta incrementi di performance significativi.

Ed ecco che, dopo questa breve analisi dei tre punti precedenti, possiamo arrivare alla prima nostra osservazione: il ridimensionamento di qualsiasi dimensione della rete (larghezza, profondità o risoluzione) migliora l'accuratezza, ma il guadagno di precisione diminuisce con modelli sempre più grandi.

Sebbene sia possibile manipolare arbitrariamente due o tre dimensioni alla volta, il ridimensionamento arbitrario è piuttosto dispendioso, ed è difficile quindi raggiungere un punto di equilibrio. La maggior parte delle volte, questa variazione manuale determina un'accuratezza ed un'efficienza non ottimali.

L'intuizione afferma che all'aumentare della risoluzione delle immagini, anche la profondità e la larghezza della rete dovrebbero essere aumentate. Campi ricettivi più ampi possono acquisire funzioni simili e sempre più complesse che includono più pixel in un'immagine. È fondamentale bilanciare tutte le dimensioni di una rete durante il ridimensionamento delle CNN per ottenere una maggiore precisione ed efficienza; EfficientNet è proprio nata per raggiungere tale obiettivo.

2.10.4 SqueezeNet

SqueezeNet è il nome di una deep neural network per la visione artificiale che è stata rilasciata nel 2016. Questa è stata sviluppata da ricercatori della DeepScale, dell'Università della California, Berkeley e della Stanford University. Il progetto di questa rete ha come obiettivo quello di creare una *rete neurale* con un numero inferiore di parametri rispetto a quelle già in circolazione, capaci di poter essere eseguite in macchine a bassa memoria, e che potesse essere più efficacemente trasmessa su una *computer network*. E' stato dimostrato [13] in questi anni che questa nuova rete ha ottime prestazioni anche in esecuzione su piattaforme di elaborazione a bassa potenza come smartphone, FPGA e processori personalizzati. Le strategie di design si possono dividere in tre regole ben precise [13]:

- **Sostituire filtri 3x3 con filtri 1x1** Dato un certo numero di filtri di convoluzione, possiamo scegliere di rendere la maggior parte di questi filtri 1x1. Un filtro 1x1 ha meno parametri rispetto a un 3x3, quindi in conclusione avremo reti meno pensanti.

- **Ridurre i canali di input a filtri 3x3** Considerando uno strato di convoluzione composto interamente da filtri 3x3, la quantità totale di parametri in questo livello è: (numero di canali di ingresso) x (numero di filtri) x (3 x 3). Possiamo da qui ridurre il numero di canali di input su filtri 3x3 usando i livelli di compressione [13].

- **Esegue il downsample in profondità nella rete in modo che i livelli di convoluzione iniziali abbiano mappe di attivazione di grandi dimensioni**

In conclusione, le strategie 1 e 2 riducono in maniera opportuna la quantità di parametri di una CNN mentre si tenta di preservare l'accuratezza con la terza. Tutto questo mira a massimizzare l'accuratezza con un *budget* limitato di parametri.

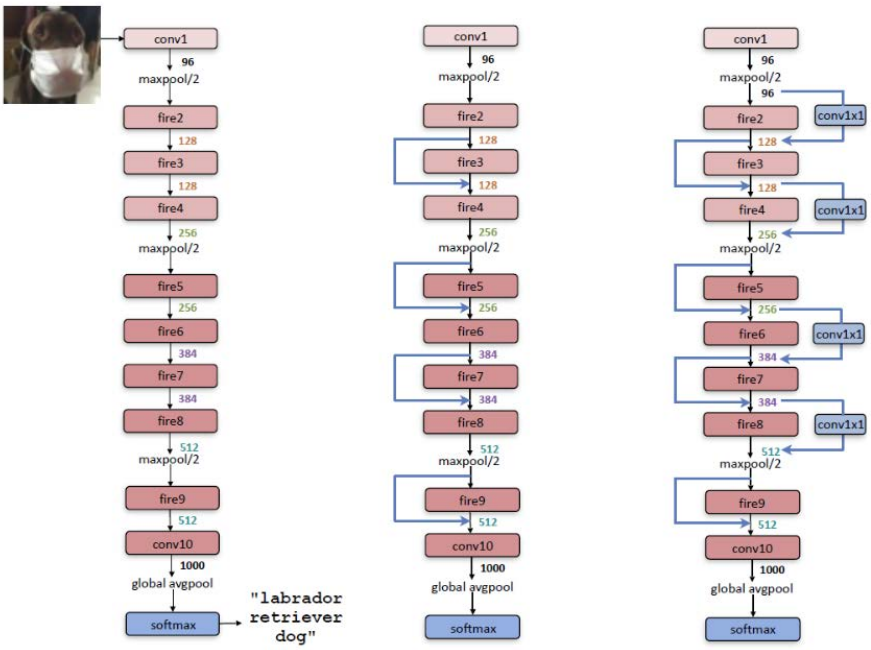


Figura 2.10: Struttura SqueezeNet [13]

Capitolo 3

Risultati precedenti e primi esperimenti

3.1 Introduzione

In questo capitolo inizialmente daremo un breve sguardo ai risultati precedentemente ottenuti con GoogleNet per poi volgerlo a possibili tecniche di *transfer learning* e allo studio di nuove reti che hanno preso piede all'interno della comunità scientifica in questi ultimi anni.

3.2 Classificazione discontinuità: risultati precedenti

Con GoogleNet l'analisi è stata svolta con circa 70 tracce video a 7,5 e 15 inch/s, e con questi è stato creato un unico dataset formato dalle immagini dei sotto-frame relativi all'area del nastro. All'epoca era stato scelto di non dividere i dati a seconda della velocità, in quanto la differenza tra i frame dei due dataset non è così incisiva.

Le classi scelte per il nuovo dataset sono **giunte, ombre e marche**. Classificando manualmente le immagini, si è ottenuto un nuovo dataset, utilizzabile poi per la fase di training, suddiviso come mostrato nella tabella seguente.

Tabella 3.1: Composizione Dataset per il training

Classe	Numero elementi
Giunte	791
Marche	700
Ombre	659

Sono stati anche estratti, da video differenti, altri 300 frame (100 per ogni classe) e questi sono stati destinati a far parte del *testSet*, per giudicare il comportamento della rete su immagini provenienti da video non noti alla rete. I rimanenti dati sono stati divisi in *trainingSet* (80%) e *validationSet* (20%), come mostrato in tabella 3.2. Per creare i tre set si e cercato di utilizzare il più possibile video diversi tra loro, con l'obiettivo di creare un dataset rappresentativo della realtà presa in esame.

Tabella 3.2: Divisione degli elementi del nuovo dataset in training, validation e test set

Training Set	Validation Set	Test Set	Totale Elementi
1720	430	300	2450

Come riportato nel documento [4], l'accuratezza ottenuta sul *validationSet* per GoogleNet si aggirava attorno al 93%. I risultati ottenuti sul *testSet* erano altrettanto buoni, indicando che la rete riusciva ad apprendere in maniera efficacie la differenza tra le varie classi su video differenti a quelli dati come input per il *training*. Questo dato dal fatto che molto probabilmente, nel dataset, era presente un campione rappresentativo per ogni classe.

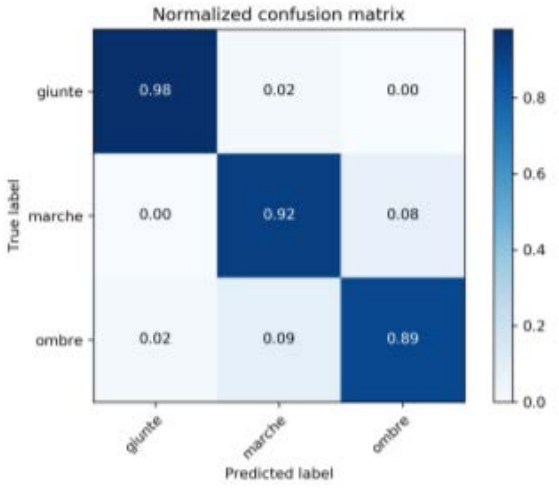


Figura 3.1: GoogleNet matrice di confusione

Da queste considerazioni si notano delle ottime performance per quanto riguardo l'allenamento di questa rete; con sole 50 epoche era possibile estrapolare *features* interessanti e capaci di riconoscere, in media nel 93% dei casi, la classe di appartenenza di un'immagine inserita come input. L'unico dubbio da considerare era la dimensionalità di GoogleNet. Nei capitoli successivi discuteremo la possibilità di utilizzare reti sempre più piccole di dimensione, cercando di individuare un ***lower bound*** per quanto riguarda *loss* ed *error rate*, soprattutto in relazione al *testSet*.

3.3 Esperimenti

3.3.1 TrainingSet e ValidationSet

Per capire al meglio il problema da affrontare, riprodurre in maniera più coerente possibile gli esperimenti con le nuove reti e per riuscire poi a confrontarli con i test condotti sulla vecchia rete neurale (GoogleNet), si è usato lo stesso Dataset di partenza. Quindi l'obiettivo che ci si è

posti è ancora quello di riuscire a classificare, in maniera più precisa possibile, le 3 tipologie di discontinuità: *giunte, ombre e marche*. La totalità delle immagini sono state divise in *trainingSet* e *validationSet* secondo la regola 80-20, tramite la funzione della libreria **Sklearn** con il comando *train test split*. Se si volesse comunque riprodurre l'esperimento in maniera tale da studiare i vari risultati, grazie al parametro *random state*, si possono ottenere sempre le stesse sequenze randomizzate. Sempre per tale motivo, le 100 foto estratte da ogni classe e la divisione del *dataset* rispecchiano perfettamente l'esperimento condotto nei lavori precedenti.

3.3.2 Pre-processing

Dopo aver creato i tre set su cui lavorare, è opportuno effettuare delle operazioni di *pre-processing* delle immagini da dare poi come input alla rete neurale. E' opportuno eseguire questo per tutti e tre i set, per mantenere un'elevata consistenza. Dato che la dimensione richiesta in input dalle varie reti è (224, 224, 3), il **resize** delle immagini è la prima cosa necessaria da fare. Si passa poi per la normalizzazione dei dati con la quale è stato dimostrato che si può infatti aumentare le performance delle reti. Come riportato nel codice sottostante un esempio utile potrebbe essere convertire ogni immagine in rappresentazione *floating point* a 32 bit, per poi dividere ogni valore del tensore per 255, in modo tale da ottenere matrici di valori compresi tra 0 e 1.

```
for img in X_training:
    img.astype('float32')
X_training = np.stack(X_training, axis=0)
X_training = X_training/255

for img in X_val:
    img.astype('float32')
X_val = np.stack(X_val, axis=0)
X_val = X_val/255
```

3.3.3 Transfer Learning

Come spiegato in precedenza, il primo obiettivo del mio elaborato è stato quello di osservare il comportamento del *training* su tale dataset, a livello di prestazioni, su reti già pre-allenate con ImageNet. Questa operazione è stata svolta in due maniere diverse:

- allenando solo l'ultimo *layer*, mantenendo quindi costanti i pesi dei livelli precedenti.
- allenando tutta la rete, mantenendo *allenabile* ogni singolo livello (i pesi di partenza non vengono inizializzati in maniera casuale o con valori nulli, ma vengono presi in considerazione quelli della rete già allenata su ImageNet).

In entrambi i casi comunque è necessario sostituire l'ultimo *layer* inserendone uno che faccia al caso nostro, ovvero solo con 3 classi di output e in grado quindi di classificare: marche, giunte e ombre.

L'idea è quella di mantenere un *GlobalAveragePooling2D* e un livello di *DropOut(0.4)* prima del livello d'output, mantenuto dai lavori precedenti, per ridurre la dimensionalità della rete. Con un ciclo sui *layer* poi, a seconda del caso, si decide quali di questi rendere *allenabili* o meno, come raffigurato nel codice sottostante.

Per entrambi gli esperimenti, il training sui dati è stato quindi eseguito con le impostazioni riportate in tabella 3.3; la dimensione della *batchSize* è stata impostata seguendo la capacità della GPU. Più alta era quest'ultima, più avremmo avuto bisogno di spazio sulla scheda NVIDIA per allenare l'intero modello. Date le dimensioni limitate delle risorse a nostra disposizione, abbiamo deciso di allenare le varie reti sulla CPU. Il valore del *tasso di apprendimento*, come accennato in precedenza, è molto spesso preferibile inizializzarlo ad 0.001 per poi decrementarlo tramite una moltiplicazione per una costante. Questa pratica infatti deriva dal tentativo di ridurre "potere di apprendimento" della rete quando il modello si sta avvicinando sempre più all'ottimo, per non saltarlo. Infatti, già con i primi esperimenti questa pratica si è dimostrata molto efficace.

Tabella 3.3: Parametri di allenamento

Parametri	Valori
Numero epoche	50
Batch size	40
Learning rate iniziale	0.001
Costante moltiplicativa	0.96
Cadenza diminuzione LR	8
Ottimizzatore	SGD

Alla fine di questa fase, l'architettura del modello viene salvata in un file JSON, mentre i pesi che la rete ha assegnato vengono salvati in un file *model.h5*; entrambi tali file verranno poi utilizzati e quindi caricati in fase di test, prima di eseguire la predizione sui dati.

3.4 Risultati

Come accennato precedentemente, per ogni rete, abbiamo svolto due prove: una allenando solo l'ultimo *layer*, l'altra rendendo *trainable* tutti i livelli della rete neurale presa in considerazione. Nei paragrafi sottostanti discuteremo le performance dei *training* per ogni rete e le varie decisioni successivamente adottate.

Inizialmente è stato pensato che, nel caso di allenamento solo dell'ultimo livello, dato che i primi esperimenti mostravano un lento decremento della *validation function* durante il *training* del modello, fosse meglio addestrare il modello già nelle prime fasi con più di 50 epoche. In questo caso infatti, come punto di partenza, è sempre stato attuato il training con 1000 epoche, per poi ridurlo osservando l'andamento della *validation loss*. Se questa non scendeva e rimaneva costante, l'esecuzione del modello veniva interrotto.

```
#example with EfficientNetB0

base_model = model = EfficientNetB0(classes=1000, include_top=False,
weights='imagenet')

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.4)(x)
predictions = Dense(3, activation='softmax')(x)


#this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

#-------------------------------------------------
#if you want all layer trainable

   for layer in base_model.layers:
        layer.trainable = true

#-----------------or--------------------------------
#if you want only last layer trainable (which were randomly initialized)

   for layer in base_model.layers:
        layer.trainable = false

#compile the model

model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])

#-------------------------------------------------
```

Figura 3.2: Esempio su come importare o rendere allenabili o meno i vari *layer* [13]

3.4.1 MobileNet e MobileNetV2

Le prime reti che abbiamo voluto analizzare nel nostro percorso sono state le MobileNets. Rispetto ai lavori svolti in passato ci si è posta una domanda fondamentale, ovvero se una rete più leggera di GoogleNet, in termini di numero di pesi, avrebbe comunque ottenuto le stesse prestazioni dopo la fase di *training*. Inoltre ci siamo chiesti se fosse possibile utilizzare una rete allenata precedentemente su ImageNet per acquisire dati e caratteristiche interessanti per poi essere trasferite per l'apprendimento nel problema preso in considerazione.

Durante le varie sperimentazioni si è notato subito come il numero di epoche necessario ad allenare la rete nel caso di *"training su ultimo livello"* fosse diverso rispetto a quello dove lo script si si trovava ad allenare l'intera rete. Nel primo caso la *loss function* scendeva molto più lentamente, tanto da farci capire che sarebbero servite circa 600/700 epoche per non aver più un apprendimento effettivo della rete. Dopo i primi esperimenti si è notato come, tenere tutti i pesi dei livelli interni della rete immutati, fosse una tecnica troppo restrittiva e insufficiente. A lungo andare, durante l'allenamento la rete tendeva a classificare tutte le immagini sotto la stessa classe tanto da portare il *tasso d'errore* a valori non considerabili. L'*error rate* è l'indice che sta ad indicare quanto la classificazione dei dati predetta dal modello si discosta dalla *label* effettiva.

Tabella 3.4: Risultati Transfer Learning solo ultimo layer

Rete	error rate
MobileNet	0.5280
MobileNetV2	0.5181

Il nostro set di dati, oltre ad essere piccolo, è molto diverso da quello originale (ImageNet). In queste condizioni potrebbe essere sconsigliabile non addestrare il classificatore sulla parte iniziale della rete, la quale contiene prevalentemente più **caratteristiche specifiche del dataset**. Dato, inoltre, che le reti analizzate sono molto pesanti, poter alterare soltanto una minima percentuale dei pesi dell'intera rete potrebbe non essere sufficiente alla classificazione e quindi sarebbe ideale rendere *"trainable"* qualche livello in più della rete.

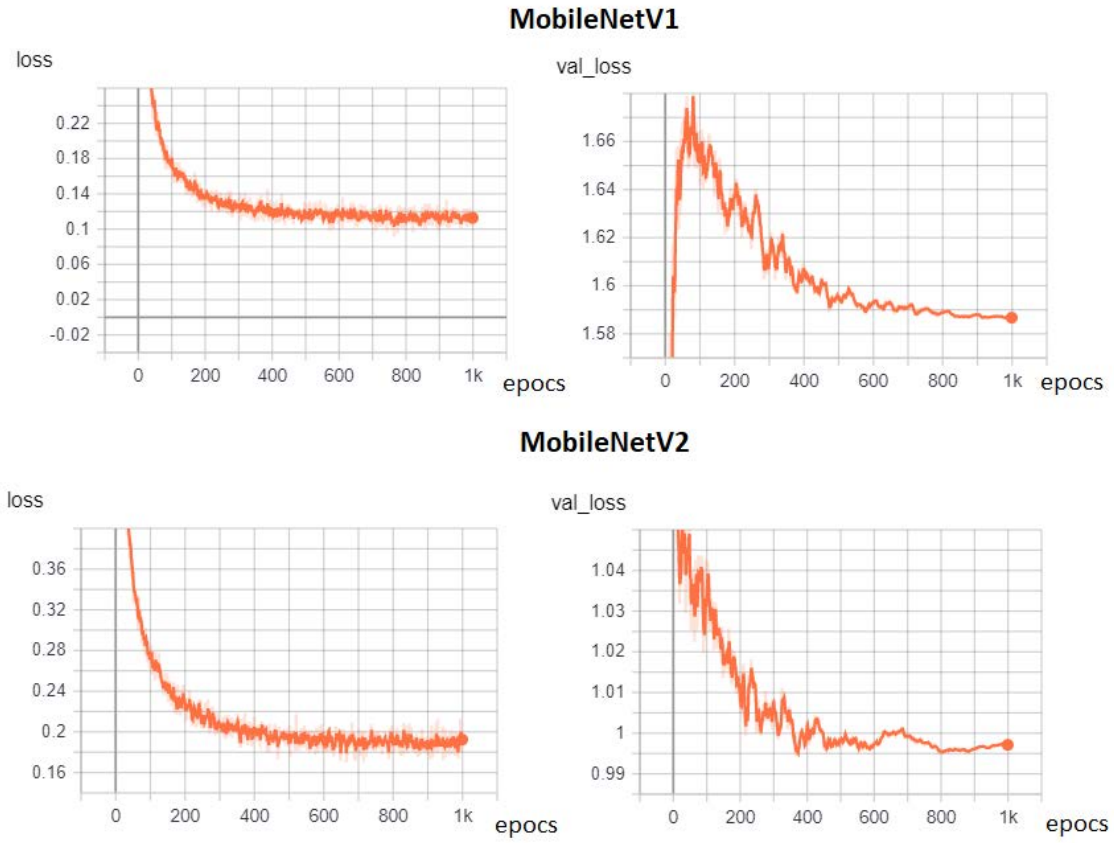


Figura 3.3: Prestazione MobileNets con training solo su ultimo *layer*

Nel caso di *transfer learning* solo su ultimo *livello*, avere una *batchSize* troppo piccola rispetto al dataset preso in considerazione, portava inoltre a delle irregolarità visibili nella curva di apprendimento (figura 3.3). Queste increspature probabilmente nascono quando, per ogni epoca, si decide di analizzare l'intero dataset suddividendolo in gruppi di campioni di piccole dimensioni, non dando la possibilità al modello di estrarre *caratteristiche* abbastanza significative/specifiche, e quindi allenare i pesi di volta in volta in maniera opportuna. Nel caso di allenamento di tutta la rete invece, come da figura 3.4, si può notare che l'andamento della curva di apprendimento sia molto più regolare e meno sensibile alla dimensione della *batchSize*. In questa tipologia di esperimenti la rete ha molti più pesi e livelli da poter modificare, inizia a memorizzare molte più **matrici di features** significative, anche se la *batchSize* è piccola rispetto al dataset complessivo.

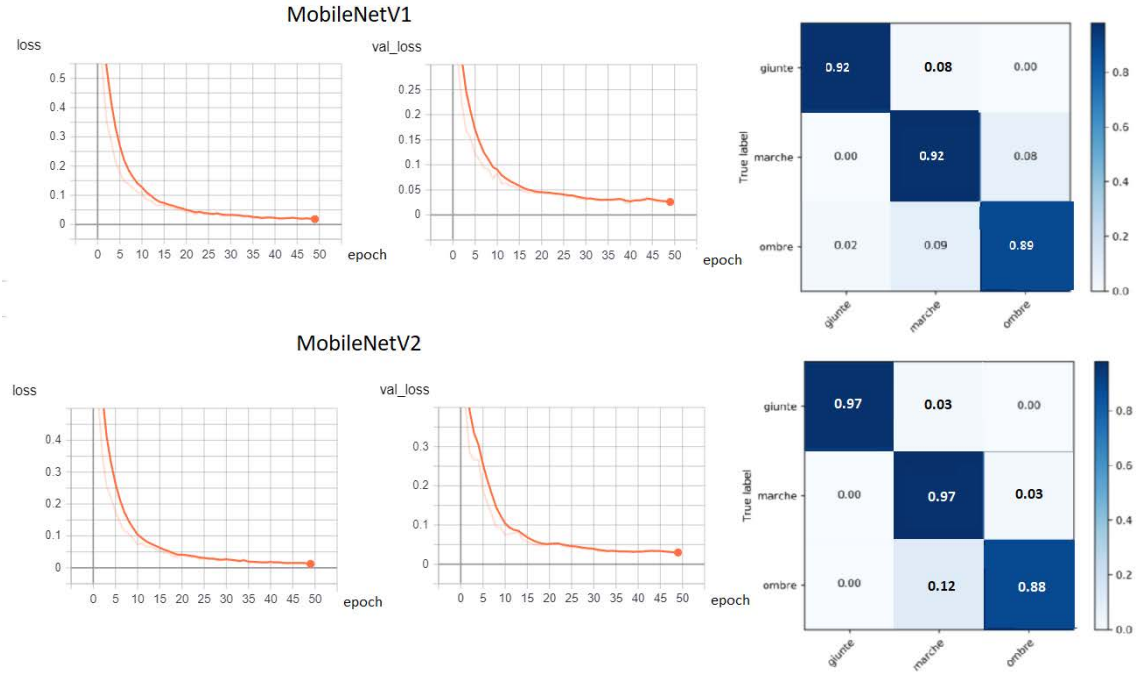


Figura 3.4: Prestazioni MobileNets con training su tutti i *layer*

Tabella 3.5: Risultati transfer learning su tutta la rete

Rete	error rate
MobileNetV1	0.0462
MobileNetV2	0.05904

Come si può notare dalla figura 3.4, gli andamenti delle due curve sono molto simili tra loro, parliamo dopo tutto di due reti aventi la stessa struttura generale. Le prestazioni in termini di *error rate*, come riportato in tabella 3.5, variano di qualche decimale, ma trascurabile. Possiamo considerare tale allenamento concluso dato che la *validation loss* inizia ad assumere un'andamento costante. Non abbiamo potuto aumentare di molto la dimensione della *batchSize* data la numerosità dei parametri interni allenabili e le dimensioni della GPU ridotte.

3.4.2 DenseNet

Dopo aver visto i pessimi risultati ottenuti su *MobileNets* nel caso del *training* esclusivamente su ultimo *layer*, ci siamo chiesti se tutto questo derivasse dalla struttura generale di questa famiglia di reti. Ci siamo domandati quale sarebbe stata la conseguenza se avessimo provato ad utilizzare un'altra rete, sempre pre-allenata su *imageNet*, ma con una complessità, in termini di densità, superiore.

Non ci fu da stupirsi quando trovammo un miglioramento di prestazioni con l'allenamento di tutta la rete (*error rate* attorno allo 3%)(fig. 3.5). Nel caso di *training solo su ultimo livello* invece, eravamo ben lontani da avere prestazioni degne di nota. In quest'ultimo caso infatti, il modello tendeva a classificare tutto sotto la stessa classe, quella di *giunta*(fig. 3.6).

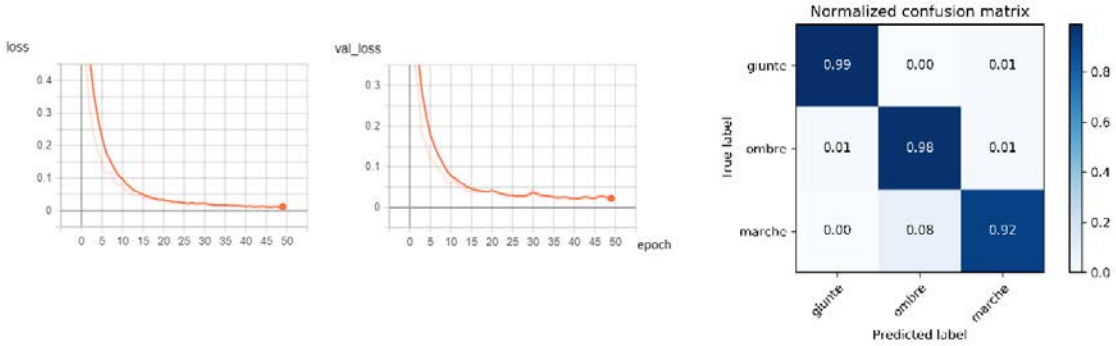


Figura 3.5: Prestazioni DenseNet121 training su tutta la rete

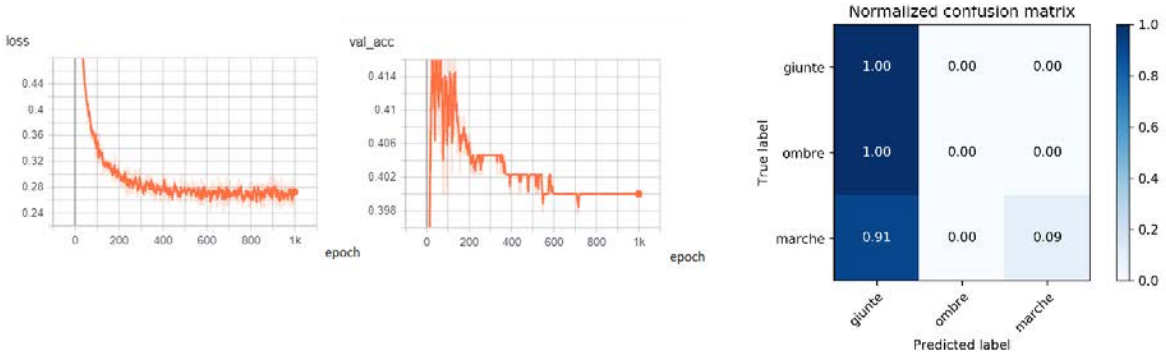


Figura 3.6: Prestazioni DenseNet121 training su ultimo layer

Data la differenza sostanziale degli andamenti dei due filoni, abbiamo deciso di abbandonare l'idea di poter allenare solo l'ultimo livello. Da qui l'esigenza e la volontà di cercare un *lower bound* di dimensionalità; ridurre quindi al minimo il numero di parametri allenabili, per osservarne poi le prestazioni.

3.4.3 EfficientNet

Date le buone *performance* ottenute con le due reti precedentemente illustrate, ci siamo chiesti se fosse possibile diminuire ulteriormente le dimensioni del modello, e riuscir quindi ad allenare un classificatore più piccolo con *error rate* comunque basso. Per il motivo spiegato nei capitoli precedenti, venne subito l'idea utilizzare EfficientNet, nata proprio per la politica del ridimensionamento. Qui notammo subito le prime diversità con le reti precedenti; non potevamo usare

più gli stessi *parametri di partenza*, e ritener l'allenamento concluso soltanto con il termine delle 50 epoche prestabilite. Perciò abbiamo provato ad osservare le prestazioni ottenute dopo 400 epoche di *training*.

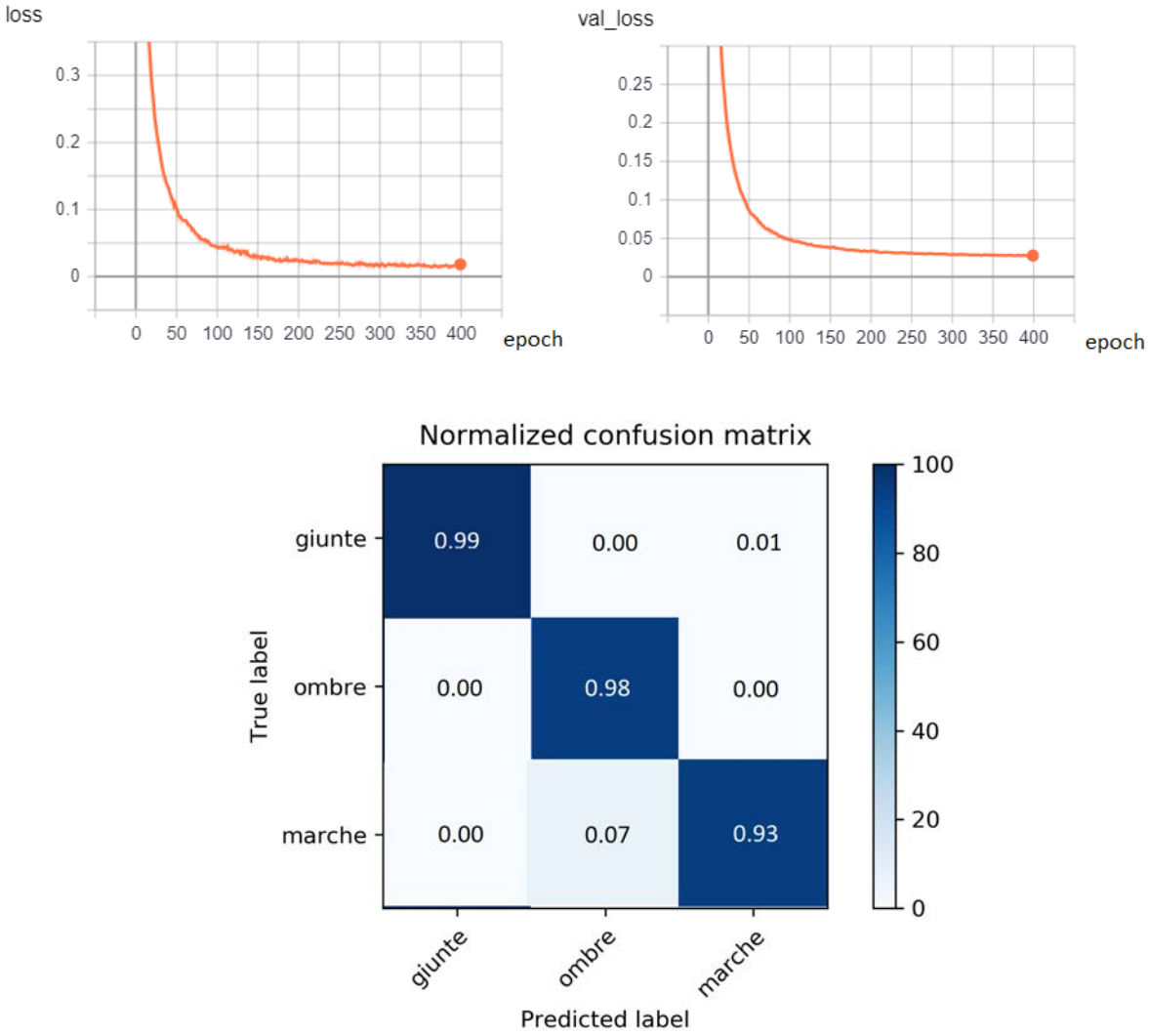


Figura 3.7: Prestazioni EfficientNet con training su 400 epoche

Dopo aver ottenuto un *error rate* del 3,3%, ci siamo chiesti se variare soltanto il numero di epoche era una tecnica che si poteva ritenere efficiente e sufficiente. Sperimentalmente si è notato che, nella maggioranza dei casi, allenare la rete nella prima fase di *training* per poi ridurre il *learning rate* nella fase di allenamento, aiuta molto spesso ad arrivare all'ottimo in maniera più veloce e sicura.

3.4.4 SqueezeNet

Con l'idea di spingerci sempre più verso reti piccole e leggere abbiamo provato ad analizzare le prestazioni ottenute allenando uno dei modelli meno complessi in circolazione: SqueezeNet.

Già con i primi esperimenti su GoogleNet, si era notato che l'utilizzo di reti che superavano il milione di pesi, era un approccio esagerato per il dataset a nostra disposizione; anzi, con *rete neurale* come SqueezeNet, si ottenevano *tassi d'errore* migliori, che si avvicinavano anche ai 2 punti percentuali sul *testSet*. L'unico compromesso che abbiamo dovuto accettare è l'aumento del numero di epoche necessarie per il *training*. Con il diminuire dei pesi infatti, per estrapolare *features* importanti, SqueezeNet necessitava di più cicli (circa 200). Il tempo totale utilizzato per l'allenamento comunque non cambiava di molto dato che, per ogni epoca, il tempo di esecuzione si era ridotto grazie alla diminuzione complessiva dei parametri.

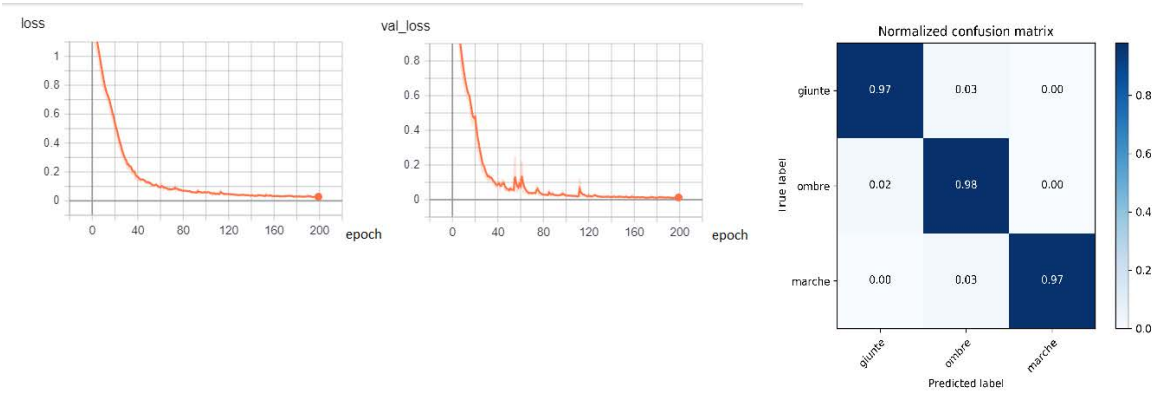


Figura 3.8: Prestazioni SqueezeNet.

Capitolo 4

Una nuova rete convoluzionale

4.1 Introduzione

Contemporaneamente allo studio delle nuove reti spiegate nel capitolo precedente, abbiamo deciso di creare una nostra *rete neurale convoluzionale* puntando a ridurre ulteriormente il peso della rete necessario per classificare il nostro dataset mantenendo comunque performance simili a SqueezeNet. Conoscendo la struttura adottata nei modelli precedenti, e le scelte fatte nel corso degli anni, si è cercato di inserire fin da subito politiche di ridimensionamento grazie i vari livelli di *pooling* e *dropOut*. Importante è notare che, in questo caso, si è usato un **ottimizzatore Adam**. Quest'algoritmo è un'estensione della *discesa stocastica del gradiente* che ha recentemente visto un'adozione più ampia per le applicazioni di *"deep learning"*, nella *computer vision* e nell'elaborazione del linguaggio naturale.

4.2 Struttura rete

MyNet è una *rete neurale* da noi creata utilizzando librerie di Tensorflow e Keras. Inizialmente abbiamo deciso di strutturare tale rete con quattro livelli convoluzionali intervallati da *Max pooling layer* utili per diminuire la dimensionalità di volta in volta. L'ultima parte della rete è costituita da un *flatten layer* e un livello di *dropOut* che genereranno le caratteristiche di input per il blocco di output della rete, ovvero una concatenazione di due *Dense layer*, l'ultimo ovviamente solo con tre classi di output. Di seguito è riportato il codice Python per costruire tale rete tramite Keras.

```
#create model

input_layer = Input(shape=(224, 224, 3))
#get number of columns in training data
n_cols = images_counter

kernel_init = keras.initializers.glorot_uniform()
```

```
bias_init = keras.initializers.Constant(value=0.2)
#add model layers
x = Conv2D(64,(3,3),padding='same', strides=(2,2), activation='relu',
kernel_initializer=kernel_init, bias_initializer=bias_init)(input_layer)
x = MaxPool2D((2, 2))(x)
x = Conv2D(64,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(128,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(128,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dense(3, activation='softmax')(x)

model = Model(input_layer, x, name='MyNetKeras')

model.compile(
  loss = 'categorical_crossentropy',
  optimizer = keras.optimizers.Adam(beta_1=0.9, beta_2=0.999,
  amsgrad=False),
  metrics = ['accuracy']
)
```

In base alla dimensione della finestra dei *livelli convoluzionali* possiamo decidere la dimensione dell'output per ogni layer della *rete neurale* e quindi il numero di pesi totali al suo interno. Abbiamo svolto tipi di sperimentazioni diverse usando rispettivamente filtri 3x3, 5x5 e 7x7. Nella tabella di seguito per ogni tipologia di rete abbiamo il numero di parametri allenabili:

Tabella 4.1: Differenze di dimensione per MyNet

Convolutional Layer	Numero di parametri allenabili
3x3	524,355
5x5	986,179
7x7	1,678,915

A differenza del capitolo precedente, dove veniva eseguito il *training* sulla rete solo dopo aver inizializzato i pesi interni con un opportuno *transfer learning* con ImageNet, qui i pesi dei modelli sono inizializzati a valori di *default*. Questi valori sono scelti attraverso la regola ***glorotUniform*** ovvero presi da una distribuzione uniforme nell'intervallo

$$[-\sqrt{\frac{6}{(fanIn + fanOut)}}, +\sqrt{\frac{6}{(fanIn + fanOut)}}]$$

dove i due parametri a denominatore rappresentano rispettivamente il numero di unità di input e il numero di unità di output del tensore di peso del layer.

E' essenziale trovare in maniera opportuna i valori dei parametri cercando di avere delle buone performance in termini di classificazione, ma anche in termini di efficienza e ridimensione. E' stato deciso di usare, come punto di partenza, gli stessi valori del capitolo 3 (*epoche, batchSize, learning rate etc.*) per il *transfer learning* per poi cambiarli di volta in volta spingendosi sempre di più verso l'ottimo.

Tabella 4.2: Parametri iniziali per il training di MyNet

Parametri	Valori
Numero epoche	50
Batch size	40
Learning rate iniziale	0.001
Costante moltiplicativa	0.96
Cadenza diminuzione LR	8
Ottimizzatore	Adam

4.3 Ottimizzatore Adam

L'ottimizzatore Adam è diverso dalla classica *stochastic gradient descent*. Quest'ultima mantiene sempre costante il *learning rate* per tutti gli aggiornamenti di peso e non cambia mai durante l'allenamento; Adam invece viene adattato separatamente man mano che si sviluppa l'apprendimento.

Questo ottimizzatore prende spunto da due differenti estensioni della *discesa stocastica del gradiente*. In particolare:

- **AdaGrad:** ha un tasso di apprendimento per parametro che aumenta e arriva al suo apice (per quanto riguarda le prestazioni) su problemi con *sparse gradients* (ad esempio tematiche di linguaggio naturale e di computer vision).

- **RMSProp:** ha tassi di apprendimento per parametro che sono adattati in base alla media delle magnitudini dei gradienti moltiplicati per ogni singolo peso. Questo algoritmo è stato proposto il conseguenza al problema di "scomparsa del learning rate" che aveva *AdaGrad* in origine.

Adam riesce a combinare in maniera opportuna sia i vantaggi di AdaGrad che di RMSProp: varia il *learning rate* in funzione alla frequenza dei campioni visionati rendendo il modello più adattivo al dataset preso in esame, e fa in modo che non questo non "scompaia" o diventi troppo piccolo in poco tempo. Nella figura sottostante si possono vedere le prestazioni dei questo ottimizzatore in confronto agli altri più utilizzati.

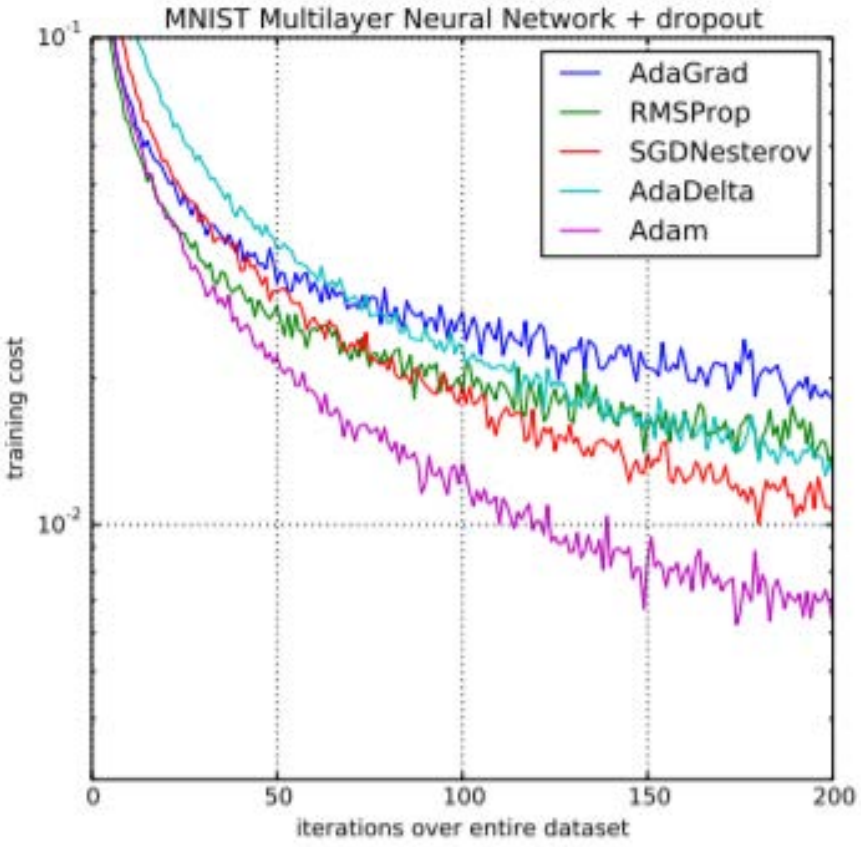


Figura 4.1: Differenze di prestazioni tra gli ottimizzatori [12]

4.4 Primi risultati

Già con i primi esperimenti si era cercato di capire se fosse necessario diramare il lavoro di ricerca in tutte e tre le configurazioni di dimensione dei filtri; si era notato che le prestazioni erano molto sibili e quindi, per rimanere in linea con l'idea di verificare che il nostro problema di classificazione di partenza, era risolvibile con una rete di piccole dimensioni, abbiamo optato per l'utilizzo dei filtri 3x3.

Allenando la rete definita precedentemente con i parametri riportati in tabella 4.2, come si può notare dall'andamento della *loss function*, essa iniziava ad assumere valore costante già dopo una trentina di iterazioni, quindi è stato deciso di ridurre il numero delle epoche necessarie per il *training*.

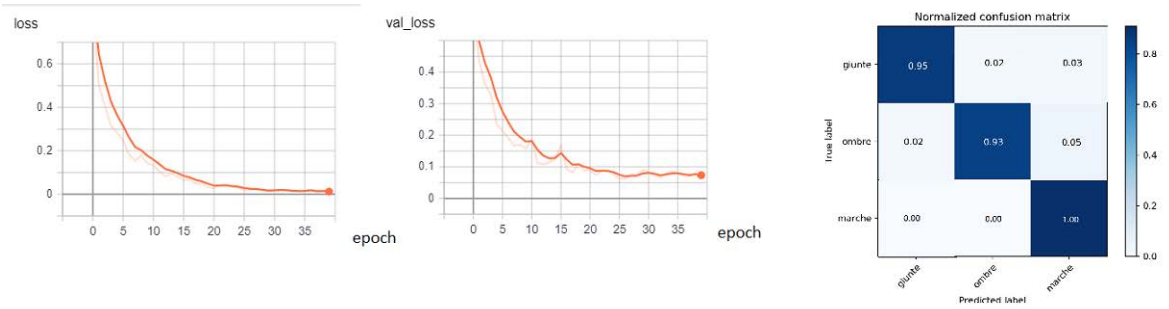
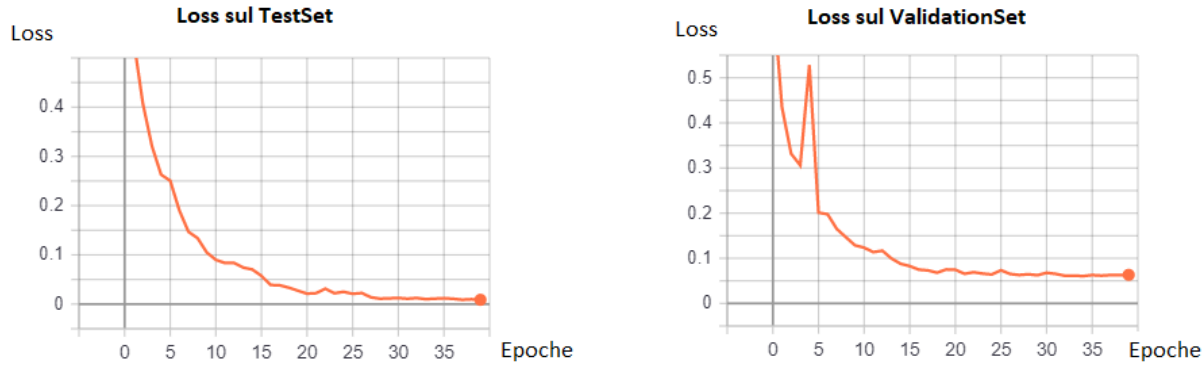


Figura 4.2: MyNet primo esperimento

Nei capitoli precedenti abbiamo spiegato perchè è meglio avere inizialmente un *tasso di apprendimento "elevato"*, e decrementarlo poi con l'aumentare delle epoche. Dato che la discesa della *loss function* nei primi cicli non è poi così elevata, si potrebbe pensare anche di variare il parametro iniziale del *drop del learning rate*, cercando di concentrare l'apprendimento della rete molto di più nella prima fase dell'allenamento per poi ridurlo drasticamente verso le ultime iterazioni. Per questo abbiamo diramato le tre sperimentazioni iniziali in altre tre diverse categorie di prove basando il tutto su tre valori di decremento diversi (0,50/0,70/0,96). Come si era intuito, le migliori prestazioni sono state ottenute nel primo caso. Di seguito riportiamo i risultati ottenuti (per il caso 0,96 figura 4.2)



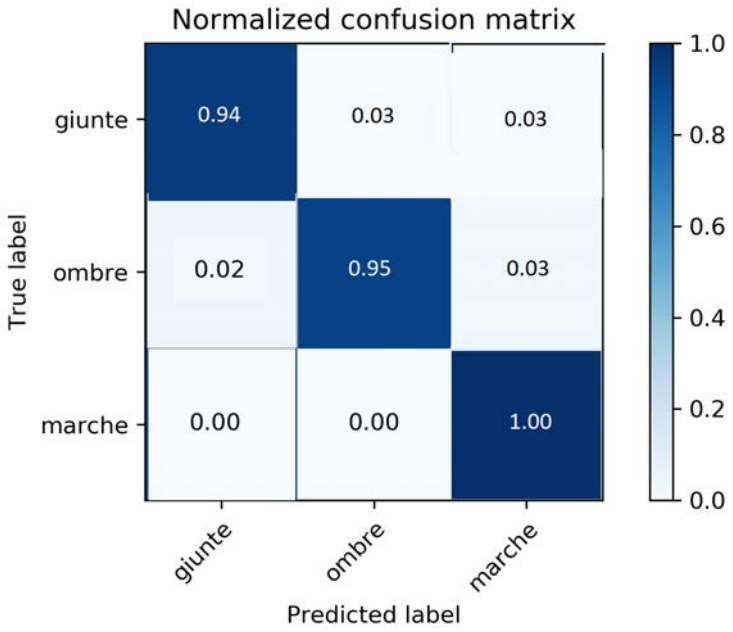


Figura 4.3: MyNet: esperimento con drop learning rate del 50%

Osservando il secondo grafico in figura 4.3, è possibile notare un'increspatura derivante molto probabilmente dalla dimensione ridotta della *batchSize*. Aumentandola infatti, sperimentalmente si è notato che la si poteva ridurre, se non eliminare. Dato che nelle epoche successive la *loss funtion* assume un andamento regolare possiamo anche sorvolare su tale considerazione, e provare a tenerlo invariato.

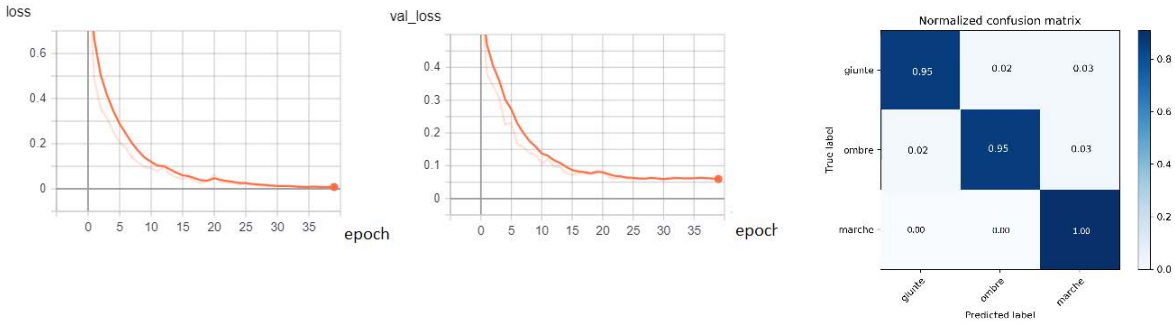


Figura 4.4: MyNet: esperimento con drop learning rate del 70%

Qui sotto riportiamo la tabella dei valori dei parametri con la configurazione migliore, relativo andamento della *loss function*, e matrice di confusione.

Tabella 4.3: Parametri configurazione migliore MyNet(3x3)

Parametri	Valori
Numero epoche	25
Batch size	20
Learning rate iniziale	0.001
Costante moltiplicativa	0.50
Cadenza diminuzione LR	8
Ottimizzatore	Adam

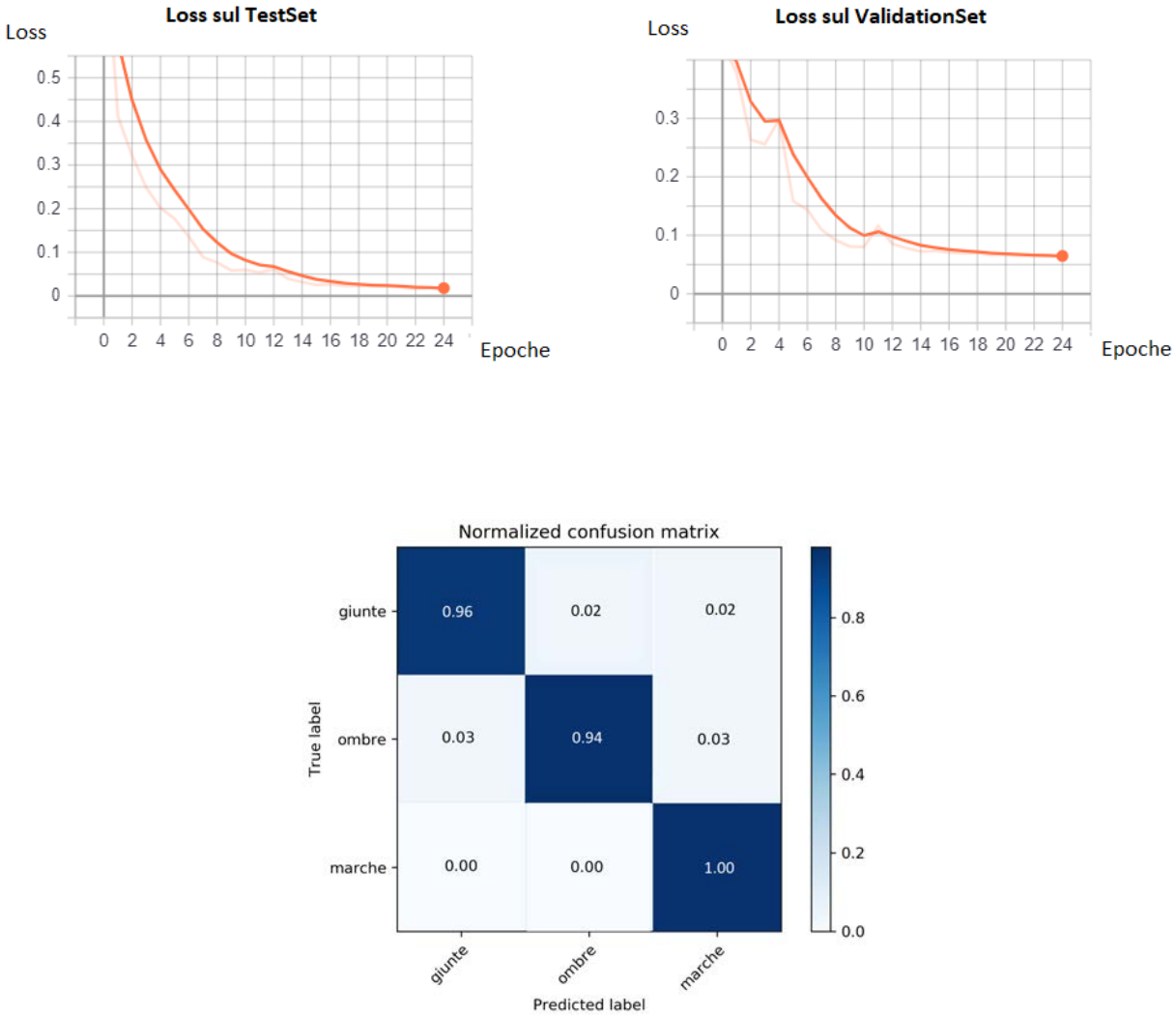


Figura 4.5: Matrice confusione miglior configurazione trovata

4.4.1 Riduzione dimensionalità

Nel paragrafo precedente abbiamo osservato le prestazioni su *test set* di una rete molto simile a SqueezeNet in termini dimensione e numero di pesi. La domanda che quindi sorge spontanea è: come cambierebbero le prestazioni riducendo il numero di *layer* rendendo così la rete meno profonda? come cambierebbero le prestazioni riducendo il numero di filtri per layer?

Di seguito vengono riportate due tabelle nelle quali verrà considerato l'*error rate* per ogni configurazione possibile in questo spazio delle possibilità. L'elemento che differenzia le due tabelle è il *drop del learning rate*; si è pensato fosse opportuno osservare anche qui la differenza delle *performance* dei due casi proprio per osservare se ci fosse o meno un miglioramento nel concentrare l'apprendimento nelle prime fasi del *training* o meno. Nelle colonne troviamo il riferimento alla configurazione relativa al numero di filtri per livello (tra parentesi l'eventuale quarto layer)

Tabella 4.4: Error rate di MyNet nelle varie configurazioni con Drop del learning rate pari 0.50

-	64-64-128(128)	32-32-64(64)	16-16-32(32)	8-8-16(16)	4-4-8(8)
3 layer	0.039	0.036	0.036	0.14	0.15
4 layer	0.036	0.033	0.039	0.049	0.12

Tabella 4.5: Error rate di MyNet nelle varie configurazioni con Drop del learning rate pari 0.96

-	64-64-128(128)	32-32-64(64)	16-16-32(32)	8-8-16(16)	4-4-8(8)
3 layer	0.049	0.046	0.10	0.14	0.11
4 layer	0.049	0.067	0.052	0.077	0.22

Come possiamo notare, allenare la rete con un *drop del learning* di 0.50, e quindi concentrare l'apprendimento nella prima fase di addestramento, porta ad un miglioramento generale delle performance. Come si poteva prevedere inoltre, le migliori prestazioni le otteniamo in media con l'utilizzo di una rete a 4 *layer* in entrambi i casi. Necessita capire se utilizzare una configurazione di rete più pesante e un miglioramento delle prestazioni minimo è un giusto compromesso tra *performance* e dimensione. Si ricorda inoltre che per ogni *training di configurazione* si ritiene l'allenamento concluso quando la *validation function* cessa di scendere. Nella tabella sottostante vengono riportate le dimensioni delle configurazioni analizzate.

Tabella 4.6: Dimensione delle configurazioni di MyNet in termini di parametri allenabili

-	64-64-128(128)	32-32-64(64)	16-16-32(32)	8-8-16(16)	4-4-8(8)
3 layer	704,451	325,603	156,915	77,755	39,471
4 layer	524,355	198,691	84,243	39,115	19,575

Come si può notare infatti dalla tabella 4.4, rispetto al caso migliore, rendendo la rete ancora più piccola (portandola a 39115 pesi allenabili) le performance sono ancora degne di nota, peggiorando di meno di due punti percentuali rispetto il caso migliore.

E' opportuno anche osservare che andando ad eliminare un ulteriore *layer* (MyNet con due soli *layer* convoluzionali) rendiamo la rete completamente inefficiente, incapace di acquisire **features** significative che probabilmente sarebbero poi servite ad identificare la classe di appartenenza dei dati inseriti come input. Portando la rete a due soli *layer* infatti la rete tende a classificare tutte le discontinuità sotto la classe di giunta.

Qui sotto vengono riportate *validation loss, loss function* e matrice di confusione delle configurazioni migliori sperimentate e precedentemente discusse.

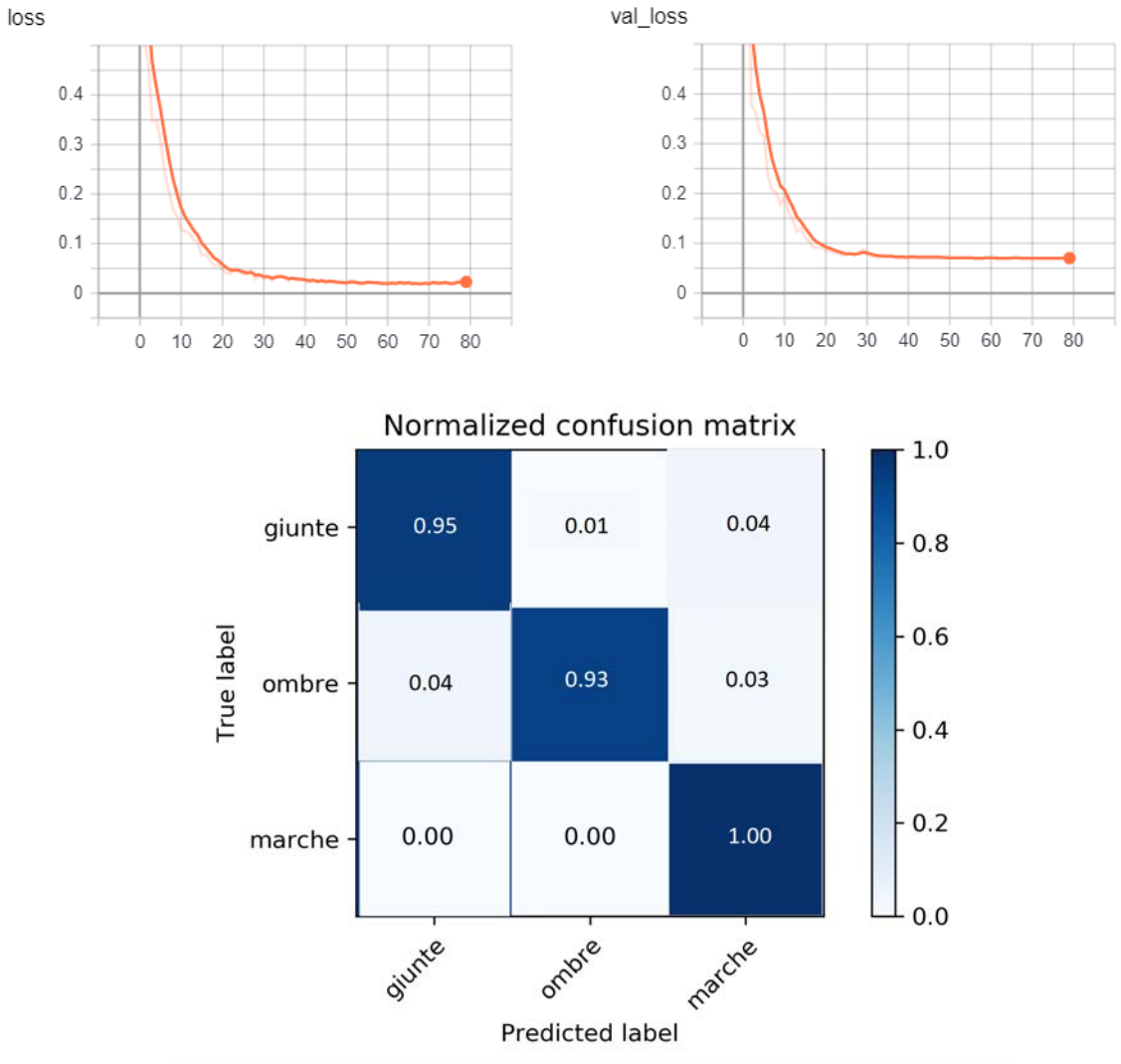


Figura 4.6: MyNet con 4 *layer* di dimensione 16/16/32/32 (DropLearningRate 0.50)

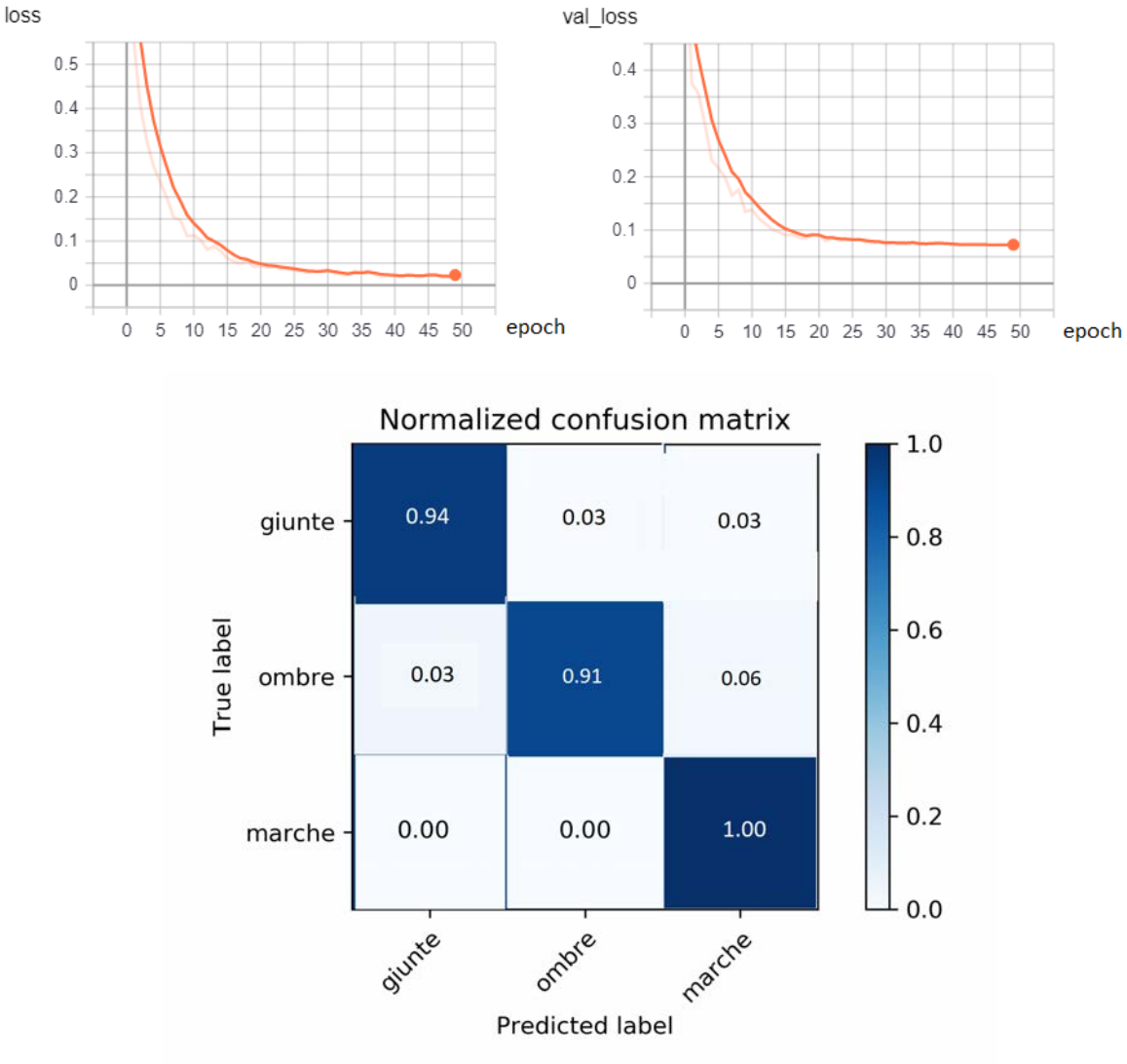


Figura 4.7: MyNet con 4 *layer* di dimensione 8/8/16/16 (DropLearningRate 0.50)

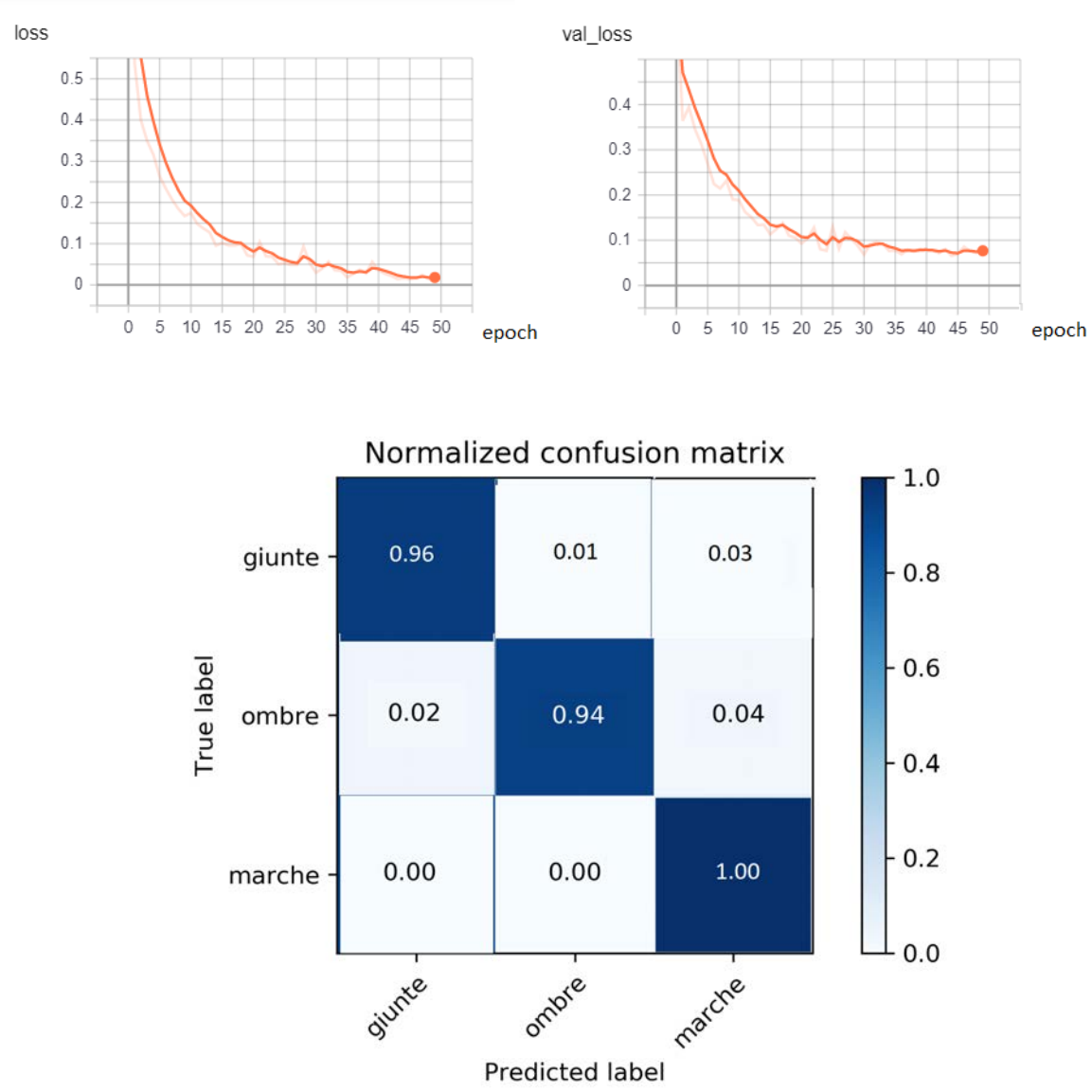


Figura 4.8: MyNet con 4 *layer* di dimensione 32/32/64/64 (DropLearningRate 0.50)

```
input_layer = Input(shape=(224, 224, 3))
#get number of columns in training data
n_cols = images_counter

kernel_init = keras.initializers.glorot_uniform()
bias_init = keras.initializers.Constant(value=0.2)
#add model layers
x = Conv2D(8,(3,3),padding='same', strides=(2,2), activation='relu',
kernel_initializer=kernel_init, bias_initializer=bias_init)
(input_layer)
x = MaxPool2D((2, 2))(x)
x = Conv2D(8,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(16,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(16,(3,3),padding='same', strides=(2,2), activation='relu')(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dense(3, activation='softmax')(x)

model = Model(input_layer, x, name='MyNetKeras')

model.compile(
  loss = 'categorical_crossentropy',
  optimizer = keras.optimizers.Adam(beta_1=0.9, beta_2=0.999,
  amsgrad=False),
  metrics = ['accuracy']
)
```

Figura 4.9: MyNet con miglior compromesso *dimensione/performance* (8/8/16/16)

Capitolo 5

Un insieme di addestramento esteso

5.1 Introduzione

In questo capitolo daremo un breve sguardo alle vecchie politiche di selezione ed estrazione, riassumendo gli aspetti fondamentali di tali procedure per poi analizzarne le problematiche. Successivamente verrà spiegata e analizzata una nuova tecnica ideata per la costruzione di un nuovo dataset cercando di risolvere gli eventuali rumori ed inconsistenze che invece erano presenti nel vecchio dataset.

5.2 Rilevazione discontinuità: risultati precedenti

Per la creazione del vecchio Dataset è stato utilizzato uno script, sviluppato in C++, con l'utilizzo della libreria OpenCV, lo script va ad analizzare un intero video, estrapolando ogni singolo fotogramma per poi elaborarlo, confrontando i frame adiacenti, per determinare la presenza o meno di una discontinuità in base alle variazioni del colore dei pixel. Il suo funzionamento e semplice; il software, una volta individuata la zona di interesse dall'utente, inserita attraverso coordinare o tramite disegno di un rettangolo sul video, va ad analizzare e comparare i frame consecutivi nell'area selezionata. Confrontando i tre canali RGB, calcola il numero di pixel che si differenziano tra le immagini prese in considerazione; se tale numero supera una certa soglia (definita come percentuale rispetto alla grandezza del rettangolo in analisi), allora il frame è considerato interessante e viene salvato, altrimenti tralasciato.

5.2.1 Selezione ROI

Nei lavori precedenti, il "problema" del *frame extraction*, è stato risolto grazie alla definizione e l'individuazione della ROI (**Region Of Interest**). L'obiettivo è quello di evitare che l'utente debba di volta in volta trovare e scrivere, nel file di configurazione di estrazione, le coordinate numeriche esatte della regione di interesse da considerare [4].

Attraverso OpenCV è stata utilizzata un'interfaccia, con cui l'utente poteva interagire e selezionare a piacimento con il mouse l'area di interesse, lasciando la gestione numerica delle

coordinate al software stesso. Sullo schermo veniva quindi visualizzato il primo frame del video preso in considerazione, nel quale tramite azione di selezione, si poteva disegnare un rettangolo per indicare al software la regione dove eseguire l'analisi dei pixel da confrontare; in caso di errori, l'area di tale rettangolo era modificabile. Una funzione di *callback* per gli eventi del mouse (*mouse handler*) gestiva la costruzione del rettangolo, in ascolto sulla specifica finestra del frame. L'analisi poteva partire una volta svolto tale procedimento per ogni singolo video.

In un secondo momento ci si è accorti che alcuni nastri non erano perfettamente allineati rispetto al piano orizzontale. Per rendere l'analisi più precisa si è quindi introdotta la possibilità di ruotare la ROI sull'interfaccia stessa: più il rettangolo è calibrato sulla dimensione e sulla posizione del nastro, più l'analisi dei pixel condurrà a risultati precisi (figura 2.8). L'utente poteva attivare l'inclinazione del rettangolo tramite il tasto *Ctrl* e ampliare o diminuire l'angolo di rotazione tramite trascinamento del mouse.

Figura 5.1: Diversi casi di selezione della ROI

Uno dei problemi principali di questo approccio era la variabilità delle dimensioni del rettangolo, in quanto lasciate da decidere all'utente. Le riprese di video diverse (distanza, angolatura) non portavano ad una standardizzazione delle dimensioni e delle velocità di riproduzione. Era inoltre necessario disegnare la ROI in modo tale che non fosse troppo piccola: la distanza percorsa dalla discontinuità tra due frame contigui poteva essere infatti maggiore della larghezza del rettangolo. Conoscendo tutte queste problematiche e sapendo che l'unica grandezza costante nei vari video era quella della testina (3 cm), per ogni velocità possiamo ridimensionare la ROI con delle semplici proporzioni inch/pixel/cm in modo tale da ovviare a tali problemi, rendendo possibile quindi anche la rilevazione di discontinuità su due frame consecutivi. Dopo una serie di test su nastri e video differenti, si notò che tramite questa semplice proporzione si era in grado di prendere una ROI consona con il video preso in considerazione, e quindi costruire il dataset.

Durante i vari esperimenti, in un secondo momento, ci si accorse che neppure queste modifiche, che sembravano essere efficaci, erano sufficienti. Come detto in precedenza, alcuni video di nastri non sono perfettamente allineati rispetto al piano orizzontale e per questo una semplice rotazione della ROI portava del rumore sulla regione selezionata. Questa tecnica infatti avrebbe preso in considerazione, come parte del nastro, anche parte dell'ambiente di riproduzione (magnetofono, sfondo, testina). Si è deciso quindi, con questo progetto, di costruire anche un nuovo Dataset da utilizzare in futuro per il *training* sui modelli che verranno studiati. La relative regole di costruzione e composizione, e le varie considerazioni verranno spiegati poi nei paragrafi successivi.

5.2.2 Gestione marche

Un ulteriore problema con cui i miei colleghi si sono dovuti scontrare nei primi anni, è la presenza di marche sul nastro. In alcuni bobine infatti, dato che queste scritte di fabbricazione si potevano presentare su ogni frame considerato, il software si trovava a rilevare tutto il nastro come una discontinuità (in questi casi la soglia di pixel diversi tra due fotogrammi consecutivi veniva infatti superata). Per questo motivo si è introdotto un controllo ulteriore sui frame destinati al salvataggio in memoria. In fase di configurazione (impostazione della soglia e selezione del video da analizzare) veniva chiesto all'utente di indicare se sul nastro erano presenti marche in maniera continua. Se presenti, il software attivava un secondo controllo sui frame prima selezionati, con il quale considerava la media del colore presente all'interno della ROI (media aritmetica del colore dei 3 canali RGB per ogni pixel). Se il colore di due frame contigui era simile in media, l'immagine non veniva salvata, altrimenti veniva considerata come discontinuità e si procedeva regolarmente a salvare il fotogramma.

L'attivazione di tale controllo si pensava fosse utile anche nel caso di video con molte variazioni di luce, il quale avrebbero portato al salvataggio di numerosi frame con ombre. Tale metodo non viene tutt'ora però utilizzato, dato che si rischierebbe di non salvare frame con scritte, segni o frammenti di nastro rovinato, la cui media di colore rimane stabile rispetto a quella dell'intero video (come accade appunto nel caso delle ombre). Altre considerazioni, più approfondite, ed esperimenti per confermare tali ipotesi si possono trovare riportate in [4]

5.2.3 Gestione numero di frame estratti

Come spiegato precedentemente, si può intuire che questo approccio è incline ad errore; il *software* si trova a salvare numerosi frame per una stessa discontinuità, poichè questa viene rilevata in posizioni diverse del nastro (in ingresso della testina, in posizione centrale e in uscita da questa). Tale problema deriva dalla velocità di riproduzione, più lento è lo scorrimento infatti, più frame della stessa giunta venivano catturati. Tutto ciò è anche aggravato dal fatto che l'analisi del video avviene ad ogni frame e, poichè il video ha un *frame rate* di 25 frame al secondo, vengono estratte immagini ogni 40 millisecondi. Questo conduce a un dataset molto rumoroso, con moltissimi frame di scarso interesse, poichè molti si riferiscono alla stessa discontinuità. Si è pensato quindi calcolare di volta in volta il numero di pixel differenti tra due frame contigui ma, una volta decretata la presenza di una discontinuità all'interno della coppia, salvare soltanto

uno di questi. Per tale motivo, in fase di configurazione viene chiesto all'utente di indicare la velocità di riproduzione del nastro. Con questo *software* sono stati estratti i fotogrammi con cui è stato creato il dataset.

5.3 Nuove tecniche di estrazione e selezione

Come detto in precedenza, in una selezione della ROI esclusivamente rettangolare, ci siamo accorti che in alcuni nastri era possibile individuare erroneamente delle regioni dell'ambiente circostante (lettore del nastro) che sarebbero state poi valutate dal modello come parte di questi.

Possiamo considerare tutto questo quindi come un elemento di disturbo nella fase di *training*.

Questo rumore è solito crearsi nel momento in cui il nastro inizia a scorrere sul dispositivo; dato che viene esercitata una leggera pressione dalla testina rotante di lettura, esso sarà soggetto ad una piccola flessione, rendendolo quindi non perfettamente allineato.

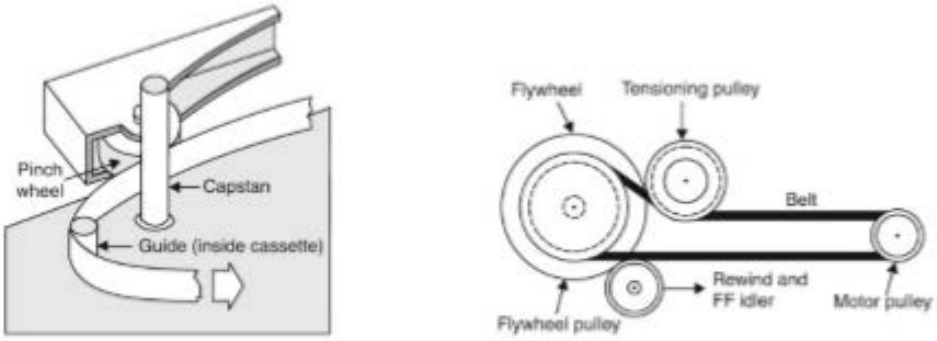


Figura 5.2: Meccanismi magnetofono [4]

Allo stesso tempo l'aggiungersi delle problematiche spieghiate nel paragrafo precedente (errori riconoscimento marche, dataset composto da frame riguardanti la stessa discontinuità ecc.), hanno obbligato a pensare che fosse quindi opportuna la creazione di un'ulteriore *dataset* con il quale allenare i vari modelli, cercando di eliminare parte di queste irregolarità. E' stato deciso quindi di non utilizzare più lo *script* di estrazione sviluppato dai mie colleghi, ma una vera e propria nuova tecnica di validazione di una *Ground Truth*, ideata all'interno del *CSC*, tramite l'utilizzo di file CSV, creati grazie ad un'individuazione manuale delle discontinuità.

Inizialmente sono stati visionati, frame per frame, 17 video differenti; su un opportuno file CSV, uno per ogni video, sono stati individuati gli intervalli temporali di tali discontinuità. Il foglio di lavoro era formato da due righe:

- Nella prima riga erano segnati i tempi
- Nella seconda riga la tipologia della discontinuità rilevata

Figura 5.3: Esempio preocedura precedente per la selezione rettangolare della ROI

00:22:04	00:24:02	00:24:05	00:25:08	00:25:12	00:25:17	00:26:23	00:27:04
Splice	Dirt	Dirt	Damage	Splice	Splice	Splice	Splice

Figura 5.4: Esempio file CSV

Dato che durante lo scorrimento del nastro, una singola discontinuità, si presenta all'interno della ROI in media per tre fame consecutivi, è stato deciso di tener conto solo dell'intervallo temporale in cui l'errore si visualizzava al centro della testina rotante. Per ridurre comunque al minimo gli errori di distrazione e interpretazione, il mio compito è stato quello di creare ulteriori due script, uno di **estrazione** e uno di **selezione**, i quali verranno spiegati nelle due sezioni successive. Al termine di entrambe le procedure, si poteva ottenere un dizionario in formato *json* dove al suo interno avremmo catalogato tutte le discontinuità rilevate e validate dei CSV di partenza, dando la possibilità all'utente inoltre di evidenziare e correggere gli eventuali errori. I due script hanno come obiettivo quello di ridurre gli effetti di bordo spiegati precedentemente, e un'individuazione sicura della discontinuità cercando di portar all'interno del dataset meno rumore possibile.

5.3.1 Nuova tecnica di estrazione

Come nei lavori precedenti si cerca di dare la possibilità all'utente di evidenziare a mano, per ogni video, la ROI; in questo caso però si cerca di ridurre al minimo la selezione dell'ambiente circostante. Si è pensato potesse essere efficace, per identificare la regione di interesse, utilizzare la solita interfaccia di selezione (OpenCV), questa volta per l'individuazione di una serie di punti di delimitazione della parte superiore del nastro e non più di un rettangolo costruito tramite trascinamento del mouse.

Con l'idea di attuare anche un controllo sul file CSV si è pensato di non salvare un unico frame per intervallo temporale, ma anche quelli che stavano in un intorno di -120/+120 millisecondi. Quindi l'utente sarà chiamato a creare una sequenza di punti, tramite un doppio click del mouse, in una finestra che apparirà appena verrà lanciato il programma, seguendo le regole riportate qui sotto:

- I primi due punti selezionati dovranno essere i vertici del lato sinistro della ROI, e serviranno per identificare l'altezza di quest'ultima. (punti 1 e 2 nella figura)
- A piacimento poi l'utente sarà chiamato a percorrere e selezionare in maniera grossolana l'intera parte superiore del nastro (punti 3,4,5...). Lo script poi userà tali *"starting point"* per calcolarsi tutte le coordinate intermedie tramite la **regola di Bresenham** e definire, pixel per pixel, l'intero bordo superiore del nastro magnetico [27,28].

```
-# "Bresenham's line algorithm" #-
def line(x0, y0, x1, y1):
    points_in_line = []
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    x, y = x0, y0
    sx = -1 if x0 > x1 else 1
    sy = -1 if y0 > y1 else 1
    if dx > dy:
        err = dx / 2.0
        while x != x1:
            points_in_line.append((x, y))
            err -= dy
            if err < 0:
                y += sy
                err += dx
            x += sx
    else:
        err = dy / 2.0
        while y != y1:
            points_in_line.append((x, y))
            err -= dx
            if err < 0:
                x += sx
                err += dy
            y += sy
    points_in_line.append((x, y))
    return points_in_line
```

Figura 5.5: Esempio nuova tecnica di selezione della ROI

Grazie l'altezza calcolata in precedenza si possono trovare poi tutte le coordinate sottostanti, individuando quindi tutti i pixel che identificano la ROI (da ricordare che anche in questa procedura viene considerata la velocità del nastro, e quindi la larghezza della ROI viene riadattata come nel lavoro precedente). Una volta fatto questo, lo script farà un *warp* creando un'immagine rettangolare eliminando così i vari rumori che erano presenti nel dataset precedente. Il warp è necessario per due ragioni:

- poter riadattare l'immagine con le considerazioni fatte ad inizio capitolo, basandosi sulla velocità del video e la dimensione della testina;
- riuscir a fare un *resize* dell'immagine perchè diventi input della rete che alleneremo successivamente.

5.3.2 Tecnica di selezione

Una volta creato l'insieme di immagini da valutare l'utente sarà chiamato a scegliere, per ogni nastro e per ogni discontinuità, una delle 7 immagini che lo script di selezione proporrà. Tramite una *widget* di *dropOut* si potrà scegliere quale nastro, e quale tipologia di discontinuità considerare nella valutazione presa in esame.

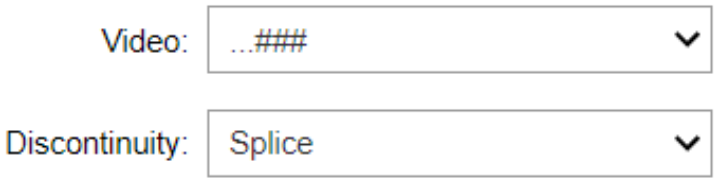


Figura 5.6: Finestra di selezione delle discontinuità

Una volta fatto questo, verrà visionata su schermo una finestra di *"selezione frame"* tramite cui l'utente sarà chiamato a scegliere quale dei frame proposti considerare tramite un semplice click del mouse (fig. 5.7). In tutti quei casi in cui l'utente non riesce a riconoscere la discontinuità nelle immagini esposte, può saltare la scelta chiudendo la finestra.

Sopra ad ogni bottone è presente un'etichetta nel seguente formato:

offsetIntorno_discontinuitàCSV_millisecondTime

Figura 5.7: Finestra di selezione delle discontinuità

Dato che c'è la possibilità che alcuni intorni temporali si *"sovrappongano"*, è opportuno segnalare all'utente se un'immagine già selezionata precedentemente è presente nelle 7 immagini presentate in quel determinato momento. Se ciò accade, il bottone non è selezionabile, e l'immagine sopra di esso sarà in scala di grigi (come si può notare nell'immagine 5.7). Una volta completato il ciclo di selezione, lo script andrà ad aggiornare un file dizionario in formato **json**, dove verranno elencate tutte le discontinuità nel seguente modo:

```
- tape_id: ...###
- tape_speed: 7.5 o 15
- tape_path (file video): //
- tape_file_name (file video): ...###.mov
- discontinuity:
                - disc_id (univoco): ...###_disc_timing
                - type: splice
                - disc_path: //
                - disc_file_name: ...###_disc_timing.jpg
                - gt_timing (ground truth - in ms): ########ms
                - disc_timing (in ms): #########ms
                - offset (in frame): -3 +3
```

Figura 5.8: Esempio frammento di dizionario relativo ad una discontinuità

Tramite un ulteriore controllo, con l'ausilio di un ulteriore file *json*, lo script è in grado di controllare quale nastro e quale discontinuità sono ancora da valutare, in maniera tale da aver la sicurezza di aver validato tutta la tabella di **Ground truth** ("0" sta per "non visionato", "1" altrimenti).

```
{
        "Name": "...333",
        "Dirt": "0",
        "Damage": "0",
        "Start": "0",
        "End": "0",
        "Splice": "0",
        "Brand": "0"
}
```

Figura 5.9: Esempio frammento del file json di controllo validazione

Al termine di questa procedura di estrazione e selezione l'utente avrà a disposizione un nuovo dataset, composto (diversamente del passato) da discontinuità prese in considerazione soltanto una volta. Questo, insieme alla riduzione del rumore, dovrebbe migliorare le *performance* dei modelli studiati.

5.4 Risultati

Dato che il lavoro di estrazione di frame manuale è una procedura lunga e incline ad errori, e dato il poco tempo a disposizione, in questo paragrafo si cercherà di evidenziare e dimostrare gli aspetti positivi dei miei due script.

Come affermato in precedenza, lo script di estrazione ha il compito di eliminare o ridurre possibili effetti di bordo derivanti dalla selezione di una ROI esclusivamente rettangolare. Con questa procedura infatti, in alcuni casi si era costretti ad includere anche la testina rotante all'interno della ROI tanto poi da risultare come "rumore" nel frame della discontinuità estratta come in figura sottostante.

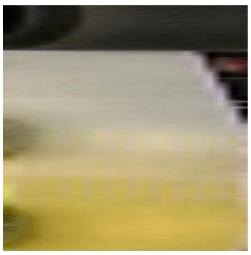


Figura 5.10: Frame estratto con vecchia procedura e con ROI rettangolare, senza warp

Tramite procedura raffigurata in 5.5, si riesce a rimuovere in molti casi questa tipologia di errore, ottenendo come risultato un frame quasi privo di effetti di bordo.

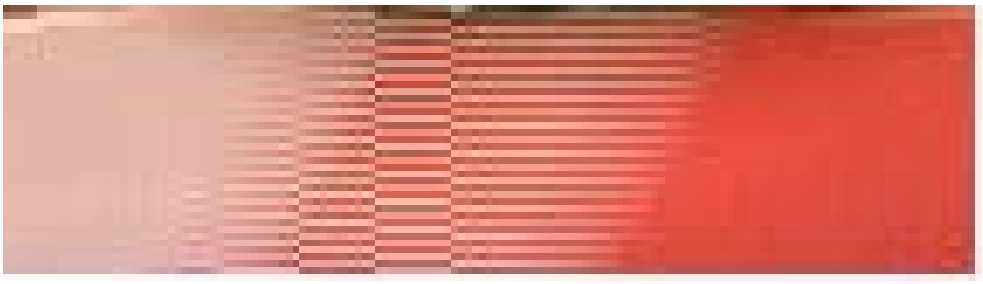


Figura 5.11: Frame estratto con nuova procedura

Non sempre questa nuova procedura di estrazione porta a dei risultati senza rumore. Per alcuni video sarà necessario ricontrollare e riformulare la ROI se lo si ritiene necessario. E' opportuno considerare comunque che, nella totalità dei casi di verifica di funzionamento degli script, e quindi in una prova fatta su una quindicina di nastri, questi rumori sono marginali e il numero totale è sensibilmente ridotto rispetto ai lavori passati.

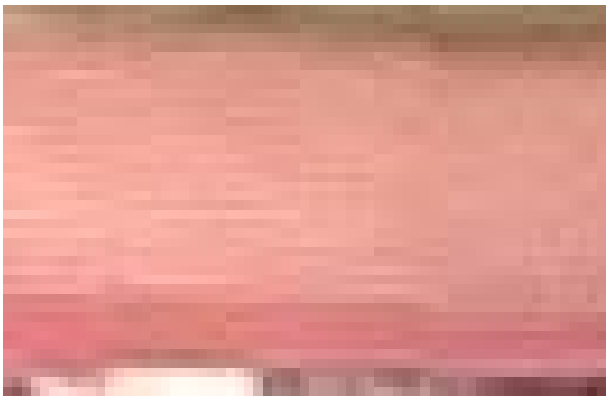


Figura 5.12: Frame estratto con nuova procedura (con rumore)

Mentre prima di estrazione precedente c'era il rischio che lo script di estrazione potesse rilevare più volte consecutivamente più discontinuità su due o più frame contigui, con questa tipologia di scelta iterativa invece, nell'intorno considerato sarà possibile estrarre e rilevare solo una discontinuità per volta. Lo script infatti, nel caso di intorni temporali che si sovrappongono, renderà eventuali frame già selezionati precedentemente non selezionabili allo step successivo, rappresentando tutto come un'immagine in scala di grigi.

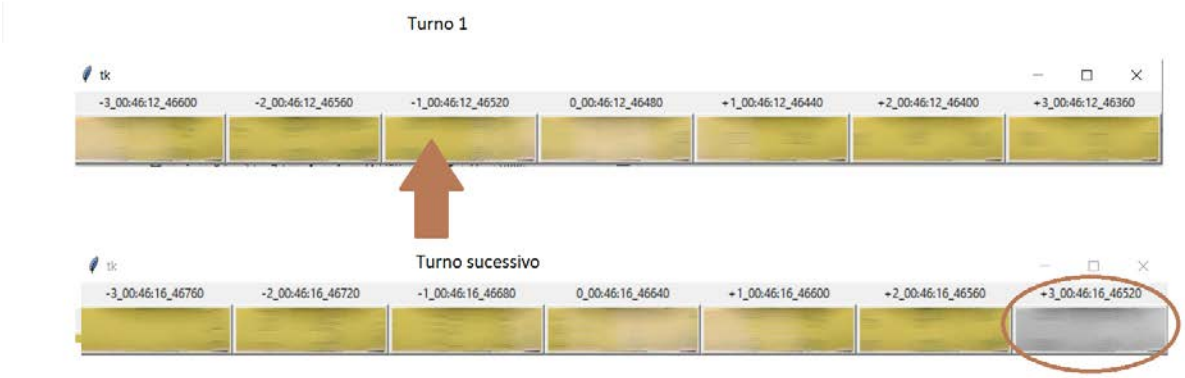


Figura 5.13: Frame estratto con nuova procedura

Con questa tipologia di selezione si può fare un'ultima considerazione: all'utente è data la possibilità di scegliere il miglior frame dell'intervallo che, una volta selezionato, potrà poi far parte del dataset, così da rendere quest'ultimo il più veritiero possibile.

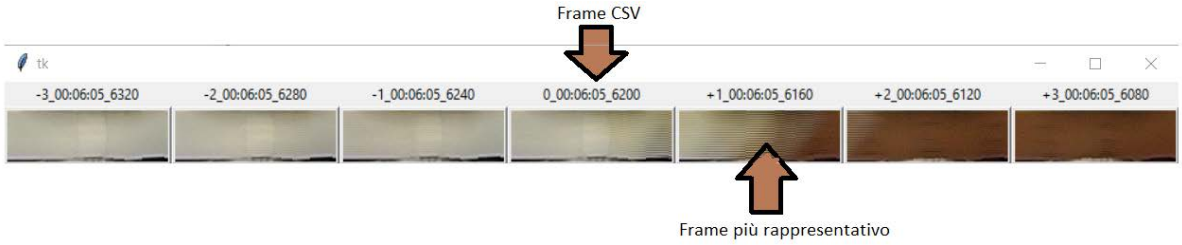


Figura 5.14: Frame estratto con nuova procedura

Ovviamente tale selezione non sarebbe sempre necessaria dato che alcune volte il frame da selezionare è effettivamente quello segnato nel CSV (ovvero quello centrale nella finestra di selezione) ma, dato che i casi in cui troviamo la discontinuità nell'intorno sono maggiori, l'uso di questa *window* sembra necessaria per avere un dataset pulito.

Capitolo 6

Conclusioni

Questa tesi si inserisce nell'ambito della conservazione e dell'accesso ai documenti sonori storici e propone degli strumenti per la l'estrazione e l'analisi delle discontinuità presenti nei nastri magnetici. Questi tool confluiranno in una collezione di software composta da interfacce di accesso [29,2,33] e algoritmi ad hoc per l'analisi e l'accesso a documenti analogici digitalizzati [30,31,32]. Il punto di partenza è stato il lavoro di diversi colleghi [3, 4] che hanno intrapreso i primi studi sull'argomento.

Il primo passo è stato il cambio di piattaforma: l'idea di sviluppare tutto il lavoro nell'ambiente di sviluppo Conda, il quale consente la creazione di environment dedicati, deriva dall'esigenza di poter sviluppare codice senza aver problemi di incoerenza tra versioni di librerie che sarebbero potute venire a galla lavorando prima nella parte di classificazione e poi di estrazione.

Durante gli anni, il lavoro dei mie colleghi, si è concentrato sull'attuazione di una procedura di rilevazione ed estrazione automatica di discontinuità tramite il confronto tra due frame consecutivi. In un secondo momento, con l'implementazione di una rete neurale, si è deciso di rendere autonoma anche la classificazione di tali frame.

Per la parte di classificazione inizialmente si era scelto di implementare una rete GoogleNet usando nuove tecnologie come Caffe. Allenando la rete su un dataset iniziale e testandola su immagini nuove e provenienti da video sconosciuti alla rete però, è emerso che il vecchio dataset non era rappresentativo della realtà presa in esame e il modello quindi stentava a generalizzare i concetti e, di conseguenza, ad imparare (all'interno del dataset erano presenti duplicati di discontinuità, frame relativi allo stesso nastro e alla stessa discontinuità). Per questo si è pensato, come parte del progetto, di creare un dataset composto da frame riguardanti più video, come validazione di una Groud Truth, in maniera tale da avere un singolo frame per ogni singola discontinuità. Tramite degli opportuni file CSV è stata costruita la nostra "tabella di verità", e grazie a dei opportuni script di estrazione e selezione (studiati in maniera tale da ridurre il "rumore" all'interno del frame che si sarebbe generato tramite selezione di ROI rettangolare) l'utente può creare ed essere sicuro di aver tra le mani un dataset per il training generalizzato.

Come ulteriore parte del mio progetto di tesi, è stato verificato che effettivamente, GoogleNet era una rete troppo pesante per il problema preso in considerazione; abbiamo osservato che anche tramite la costruzione/composizione di una piccola rete costruita grazie la concatenazione di 3/4 layer convoluzionali si potevano avere performance degne di nota.

In futuro potrebbe essere un'idea verificare se effettivamente i due script, di estrazione e classificazione, possono aiutare ad avere un dataset più consono all'obbiettivo da raggiungere. Inoltre sarebbe opportuno migliorare le prestazioni di MyNet non solo cambiando il numero dei filtri utilizzati nei *layer*, ma cercando di riuscire ad individuare delle strutture di gruppi di *layer* già presenti allo stato dell'arte ma che possano rendere le *performance* di reti così piccole migliori. Potrebbe essere opportuno inoltre rendere anche il mio script di estrazione automatizzato basandolo sulla rilevazione di discontinuità tramite il calcolo della media della variazione dei canali RGB dei pixel della ROI. Per eliminare il problema dei duplicati di discontinuità si potrebbe procedere in questo modo:

- implementare, nello script di Ylenia e Mirco, la creazione di una ROI non rettangolare tramite individuazione di punti, per poi fare un *warp* dell'immagine, come spiegato in questo elaborato.
- fare in modo che lo script di estrazione salvi e rinomini i frame rilevati in maniera tale che abbiano scritto, nel nome, *"l'intervallo temporale"* in cui è stata rilevata tale discontinuità.
- una volta create le varie directory di discontinuità in base alle tipologie, si possono dividere i frame in gruppi basandosi sugli intervalli in cui sono stati rilevati. Quindi, una volta scelto il video e la tipologia di discontinuità su cui lavorare, una volta messi in ordine crescente i frame della cartella considerata, in base al nome dato prima, si inizializza il primo gruppo; in questo insieme ci sarà il primo frame della lista e tutte le segnalazioni che seguono entro 240 ms. I gruppi seguenti si possono costruire con la medesima regola, partendo però dal frame successivo all'ultimo frame inserito nel gruppo precedente. I gruppi in questo caso non dovrebbero sovrapporsi. Una volta creati i vari insiemi, basandosi sullo script di selezione, si potrebbe far vedere a schermo all'utente, in sequenza, i vari intervalli creati e dargli la possibilità di scegliere l'immagine più rappresentativa di tale gruppo, la quale farà poi parte del dataset finale.

In questo modo il dataset finale non dovrebbe avere duplicati di discontinuità, e aver frame con meno rumore grazie la creazione della ROI non rettangolare.

Appendici

Appendice A

Codice

A.1 Script Estrazione

Importato direttamente dal sorgente Jupyter

```
## Import packages ##
import json
import math
import torch
import os
import cv2
import numpy as np
import csv
import glob
from PIL import Image
import matplotlib.image

#class for store coordinates with mouse double click
class CoordinateStore:
    def __init__(self):
        self.points = []

    def select_point(self,event,x,y,flags,param):
            if event == cv2.EVENT_LBUTTONDBLCLK:
                cv2.circle(img,(x,y),3,(255,0,0),-1)
                self.points.append((x,y))

# Bresenham's line algorithm for find ALL coordinates of the top of the ROI
def line(x0, y0, x1, y1):
        "Bresenham's line algorithm"
        points_in_line = []
        dx = abs(x1 - x0)
        dy = abs(y1 - y0)
        x, y = x0, y0
        sx = -1 if x0 > x1 else 1
        sy = -1 if y0 > y1 else 1
```

```
        if dx > dy:
            err = dx / 2.0
            while x != x1:
                points_in_line.append((x, y))
                err -= dy
                if err < 0:
                    y += sy
                    err += dx
                x += sx
        else:
            err = dy / 2.0
            while y != y1:
                points_in_line.append((x, y))
                err -= dx
                if err < 0:
                    x += sx
                    err += dy
                y += sy
        points_in_line.append((x, y))
        return points_in_line

#method for warp: with path of image and Roi coordinates define a new
matrix of pixel for define new frame for dataset
def listColor(path,coo):
    px = cv2.imread(path)
    pixelArray = []
    for i in range(0,len(coo)):
        pixelArray.append(px[coo[i][1],coo[i][0]])
    return pixelArray

#find coordinates of ROI
# A-----------------B
# |                 |
# D-----------------C

#function for define ROI with user point selection
def rect(multipoint):
    ab = multipoint
    totalAB = [] #AB side
    rect = []
    for i in range (1,len(multipoint)-1):
        temp = line(multipoint[i][0],multipoint[i][1],multipoint[i+1][0],
        multipoint[i+1][1])
        for o in range (0,len(temp)-1):
            totalAB.append(temp[o])
    totalAB.append(multipoint[len(multipoint)-1])
    tot = len(totalAB) #weigth
    h = multipoint[0][1]-multipoint[1][1] #height
    for o in range(0,h+1):
        for i in range (0,len(totalAB)):
            rect.append((totalAB[i][0],totalAB[i][1]+o))
```

```
    return rect , tot
    #method return an array with all coordinates of ROI and its weigth

#Parse CSV
import glob
for filename in glob.glob(os.path.join(path, '*.csv')): #cicle that take
                                                        all csv of the directory
    print(filename)
    video = [] #video list
    time = [] #time list
    disc = [] #discontinuity list
    split = []
    height = 0
    tot = 0
    with open(filename, 'rt') as f: #open csv file
        data = csv.reader(f)
        for row in data:
            video.append(row)
    firstRow = video[0]
    x = firstRow[0].split(';')
    secondRow = video[1]
    y = secondRow[0].split(';')
    name = y[0] # name of video
    speed = y[2] #speed of video
    #------------------------------conversion speed (ips to fps)---------------
    fps = (float(speed)*96)/224
    print("name: "+ name)
    print("speed: "+ speed)
    #--------------------------convert discontinuity time to ms---------------
    for i in range (3,len(x)):
        split = x[i].split(":")
        m = int(split[0])
        s = int(split[1])
        nframe = int(split[2])
        mseconds = (((m*60) + s)*1000)+(40*nframe)
        mseconds1 = (((m*60) + s)*1000)+(40*(nframe+1)) # +1 frame
        mseconds2 = (((m*60) + s)*1000)+(40*(nframe-1)) # -1 frame
        mseconds3 = (((m*60) + s)*1000)+(40*(nframe+2)) # +2 frame
        mseconds4 = (((m*60) + s)*1000)+(40*(nframe-2)) # -2 frame
        mseconds5 = (((m*60) + s)*1000)+(40*(nframe+3)) # +3 frame
        mseconds6 = (((m*60) + s)*1000)+(40*(nframe-3)) # -3 frame
        #repite discontinuity tipe and define frame interval
        time.append(mseconds6)
        time.append(mseconds4)
        time.append(mseconds2)
        time.append(mseconds)
        time.append(mseconds1)
        time.append(mseconds3)
        time.append(mseconds5)
        disc.append(y[i])
        disc.append(y[i])
```

```
        disc.append(y[i])
        disc.append(y[i])
        disc.append(y[i])
        disc.append(y[i])
        disc.append(y[i])
    print(time)
    print(disc)
    #capture frame on specific time
    vidcap = cv2.VideoCapture(dirVideo + name +'.mov')
    vidcap.set(cv2.CAP_PROP_FPS, fps)
    #user's ROI selection
    coordinateStore1 = CoordinateStore()
    pts1 = []
    hasFrames,img = vidcap.read()
    cv2.namedWindow('image')
    cv2.imwrite(dirVideo + "/tmp.jpg",img) #temporary image for points selection
    cv2.setMouseCallback('image',coordinateStore1.select_point)
    while(1):
        cv2.imshow('image',img)
        k = cv2.waitKey(20) & 0xFF
        if k == 27:
            break
    cv2.destroyAllWindows()
    print("Selected Coordinates1: ")
    for i in coordinateStore1.points:
        pts1.append(i)
    print(rect(pts1)[0])
    height = pts1[0][1]-pts1[1][1]+1 # height, width of ROI
    width = rect(pts1)[1]
    print(height, width)
    #cicle for frame extraction on the video
    for o in range (0,len(time)):
        tipe = disc[o] #discontinuity type
        vidcap.set(cv2.CAP_PROP_POS_MSEC,time[o])
        hasFrames,img = vidcap.read()
        cv2.imwrite(dirVideo + "/tmp.jpg",img)
        #use temporary image for ROI selection
        listRGB = listColor(dirVideo + "/tmp.jpg",rect(pts1)[0])
        #save ROI color pixel
        try:
            os.remove(dirVideo + "/tmp.jpg") #remove temporary image
        except: pass
        data=np.array(listRGB)
        data=np.reshape(data,(height,width,3))
        cm = width/3 #relation between pixel and centimeters
        if speed in ['15']:
            newWidth=cm*2.25  #resize ROI for speed Condition
        else:
            newWidth=cm*1.125 #resize ROI for speed Condition
        half=width/2
        shift=newWidth/2
```

```
        matplotlib.image.imsave(dirData + tipe +"/"+ name +"("
        + str(time[o]) + ")" +".png",data)
        img = cv2.imread(dirData + tipe +"/"+ name +"(" + str(time[o])
        + ")" +".png")
        try:
            os.remove(dirData + tipe +"/"+ name +"(" + str(time[o]) + ")" +".png")
        except: pass
        height1, width1, channels = img.shape
        crop_img = img[0:height1, round(half-shift):(round(half-shift))
        +round(newWidth)]
        height1, width1, channels = crop_img.shape
        matplotlib.image.imsave(dirData + tipe +"/"+ name +
        "(" + str(time[o]) + ")"+".png",crop_img)
```

A.2 Script Selezione Frame

Importato direttamente dal sorgente Jupyter

```
#---------IMPORT PACKAGES-----------------
import json
import math

import os
import numpy as np
import csv
from decimal import *
import glob
import cv2
import numpy as n

import matplotlib.image
import operator
import ipywidgets as widgets
from tkinter import*

#check on check.json videos that have yet to be rated
with open('check.json', 'r') as f:
    video_dict = json.load(f)

print("what is missing? ")

for video in video_dict:
    print(video["Name"])
    if video["Brand"]=="0":
        print("Brand")
    if video["Splice"]=="0":
        print("Splice")
    if video["Dirt"]=="0":
        print("Dirt")
    if video["Start"]=="0":
```

```
        print("Start")
    if video["End"]=="0":
        print("End")
    if video["Damage"]=="0":
        print("Damage")
    print("")

    #Name of video
video=['...365','...356','...332','...303']
speed=['7.5','7.5','15','15']
#Name of discontinuity
disc=['Splice', 'Damage', 'Start','End','Dirt','Brand']

#method of widget
def on_change(change):
    if change['type'] == 'change' and change['name'] == 'value':
        print(change['new'])

#video widgets dropDown
w = widgets.Dropdown(
    options=video,
    value='BERIO319',
    description='Video:',
)

#discontinuity widgets dropDown
wd = widgets.Dropdown(
    options=disc,
    value='Splice',
    description='Discontinuity:',
)


w.observe(on_change)
wd.observe(on_change)
display(w)
display(wd)

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

count = []
imageChoose = ["null"]

#callback of the botton
def clicked(arg, position):
    before = arg
    print('You clicked '+before)
    print("")
```

```
    imageChoose.append(arg)
    if position == 0:
        count.append(-3)
    if position == 6:
        count.append(+3)
    if position == 1:
        count.append(-2)
    if position == 5:
        count.append(+2)
    if position == 2:
        count.append(-1)
    if position == 4:
        count.append(+1)
    if position == 3:
        count.append(+0)
    root.destroy()

#callback for grayScale image
def skip():
    print("click another image")

#path
path = "./extracted_Frame/"+wd.value
newPath = "./newDataset/"+wd.value
imagesPath = ["null"]
filename = "./csv/"+w.value+".csv" #name of csv file that i want check
print(filename)
video = []
time = ["null"] #vector for laber of window selection
with open(filename, 'rt') as f:
        data = csv.reader(f)
        for row in data:
            video.append(row)
#split first and second rows of csv
firstRow = video[0]
x = firstRow[0].split(';')
secondRow = video[1]
y = secondRow[0].split(';')
name = y[0] #video's name
speed = y[2] #video's speed
#conversion speed (ips to fps)
fps = (float(speed)*96)/224
print("name: "+ name)
print("speed: "+ speed)
large = 0
if float(speed) == 15:
    large = 85
else:
    large = 42
#convert time to second cicle
for i in range (3,len(x)):
```

```
    if y[i] == wd.value:
        split = x[i].split(":")
        m = int(split[0])
        s = int(split[1])
        nframe = int(split[2])
        mseconds = (((m*60) + s)*1000)+(40*nframe)
        mseconds1 = (((m*60) + s)*1000)+(40*(nframe+1))
        mseconds2 = (((m*60) + s)*1000)+(40*(nframe-1))
        mseconds3 = (((m*60) + s)*1000)+(40*(nframe+2))
        mseconds4 = (((m*60) + s)*1000)+(40*(nframe-2))
        mseconds5 = (((m*60) + s)*1000)+(40*(nframe+3))
        mseconds6 = (((m*60) + s)*1000)+(40*(nframe-3))
        time.append(x[i])
        time.append(x[i])
        time.append(x[i])
        time.append(x[i])
        time.append(x[i])
        time.append(x[i])
        time.append(x[i])
        imagesPath.append(w.value+"("+str(mseconds6)+").png")
        imagesPath.append(w.value+"("+str(mseconds4)+").png")
        imagesPath.append(w.value+"("+str(mseconds2)+").png")
        imagesPath.append(w.value+"("+str(mseconds)+").png")
        imagesPath.append(w.value+"("+str(mseconds1)+").png")
        imagesPath.append(w.value+"("+str(mseconds3)+").png")
        imagesPath.append(w.value+"("+str(mseconds5)+").png")

print(imagesPath)#create an ordered vector of the image path where each group of seven i
ofset = ["+3","+2","+1","0","-1","-2","-3"]
u = 1
o = 1
while o <= (len(imagesPath)-7):
    root = Tk()
    print("prima hai selezionato: "+ imageChoose[u-1] + " stai guardando:")
    for i in range(0,7):
        if imagesPath[o+i] != imageChoose[u-1]:
            print(imagesPath[o+i])
            btn = tk.Button(command = lambda arg=imagesPath[o+i],
            position = i: clicked(arg,position))
            x = (((imagesPath[o+i].split(")"))[0]).split("("))[1] #take time in ms for l
            btn.img = tk.PhotoImage(file= path + "/"+ imagesPath[o+i])
            l1 = Label(root, text = ofset[i]+"_"+str(time[o+i])+"_"+x)
            btn.config(image=btn.img)
            btn.grid(row = 1, column = 7-i)
            l1.grid(row = 0, column =  7-i)
        else:
            btn = tk.Button(command = lambda : skip())
            #read image
            img_g = cv2.imread(path + "/"+ imagesPath[o+i])
            gray = cv2.cvtColor(img_g, cv2.COLOR_BGR2GRAY)
            #save image
```

```
            cv2.imwrite(path + "/"+ "gray.png",gray)
            btn.img = tk.PhotoImage(file = path + "/"+ "gray.png")
            try:
                os.remove(path + "/"+ "gray.png")
            except: pass
            btn.config(image=btn.img)
            x = (((imagesPath[o+i].split(")"))[0]).split("("))[1]
            #take time in ms for label composition
            l1 = Label(root, text = ofset[i]+"_"+str(time[o+i])+"_"+x)
            btn.grid(row = 1, column = 7-i)
            l1.grid(row = 0, column =  7-i)
    o = o + 7
    u = u + 1
    root.mainloop()
    print(imageChoose)

print(imageChoose)
print("")

a = []
#COMPILE JSON: DISCONTINUITY DICT
for i in range(0,len(imageChoose)):
    img = cv2.imread(path+'/'+imageChoose[i])
    resizedImage = cv2.resize(img, (224,224), interpolation = cv2.INTER_AREA)
    matplotlib.image.imsave(newPath+'/'+imageChoose[i], resizedImage)
    stringtmp = imageChoose[i].split(')')
    name = stringtmp[1]+stringtmp[0]+")"
    timing=stringtmp[0].split("(")
    time = timing[1]
    entry = {'tape_id': v,
            'tape_speed': s,
            'tape_path': 'C:/Users/Matte/discontinuity_detection_v2/frame_extraction/vid
            'tape_file_name': v+'.mov',
            'discontinuity:': {'disc_id': v+stringtmp[0]+")",
                               'type': d,
                               'disc_path': newPath,
                               'disc_file_name': stringtmp[0]+")" + '.jpg',
                               'gt_timing': int(timing[1])+(40*int(count[i])),
                               'disc_timing': timing[1],
                               'offset': count[i]
                              }}
    with open('discontinuity_dict.json', 'a') as outfile:
        outfile.write(json.dumps(entry))
        outfile.close()
```

Bibliografia

[1] Federica Bressan and Sergio Canazza. *A Systemic Approach to the Preservation of Audio Documents: Methodology and Software Tools.* Journal of Electrical and Computer Engineering, 2013

[2] Fantozzi Carlo and Bressan Federica and Pretto Niccolò and Canazza Sergio. *Tape music archives: from preservation to access*. International Journal on Digital Libraries, 2017

[3] Mirco Maniero. *Analisi automatica di nastri magnetici con reti neurali e tecniche di visione computazionale.* Tesi di Laurea Magistrale, Università degli Studi di Padova, 2016.

[4] Ylenia Grava. *Progettazione e sviluppo di un software per il riconoscimento e la classificazione di discontinuità nei nastri magnetici audio, basato su tecniche di computer vision e neural network*. Tesi di Laurea Magistrale, Università degli Studi di Padova, 2019.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand and Marco Andreetto, Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Cornell University, 17 Apr 2017

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. *Densely Connected Convolutional Networks*. Cornell University, 28 Jan 2018

[7] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. Cornell University, 4 Nov 2016

[8] Mingxing Tan, Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Cornell University, 23 Nov 2019

[9] Jason Brownleen. *A Gentle Introduction to Transfer Learning for Deep Learning.* https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Cornell University, 21 Mar 2019

[11] Stanford University. *Transfer Learning - CS231n Convolutional Neural Networks.* http://cs231n.github.io/transfer-learning/

[12] Jason Brownlee. *Adam: A Method for Stochastic Optimization*. Cornell University, 30 Jan 2017

[13] *Review: SqueezeNet (Image Classification). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. Cornell University, 4 Nov 2016

[14] *EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling*. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Cornell University, 23 Nov 2019

[15] *Understanding Neural Networks*. https://towardsdatascience.com/understanding-neural-networks-19020b758230

[16] *Reti Neurali Convoluzionali* https://www.spindox.it/it/blog/reti-neurali-convoluzionali-il-deep-learning-ispirato-alla-corteccia-visiva/

[17] Ray Edmondson. *Memory of the world: general guidelines to safeguard ducumentary heritage*

[18] Canazza Sergio, De Poli Giovanni, and Vidolin Alvise. *La conservazione dei documenti audio: un'innovazione in prospettiva storica.* Archivi, VI (2), pages 756, 2011.

[19] Niccolo Pretto, Carlo Fantozzi, Edoardo Micheloni, Valentina Burini, and Sergio Canazza. *Computing methodologies supporting automatic analysis of electroacoustic music on analog magnetic tape*. Computer Music Journal, 2019.

[20] Niccolò Pretto. *Cultural context-aware models and IT applications for the exploitation of musical heritage*. PhD thesis, Universita degli Studi di Padova, 2019.

[21] Wido Th. van Peursen, Ernst Thoutenhoofd and Adriaan van der Weel. *Text Comparison and Digital Creativity*. 29 Oct 2010

[22] Smith, Abby. *Strategies for building digitized collections, "Microform and imaging review"*. 1 Mar 2002

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going deeper with convolutions. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* June 2015.

[24] Deep Learning for humans, *https://github.com/keras-team/keras*

[25] OpenCv, *https://opencv.org/*

[26] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun, *Deep Residual Learning for Image Recognition*. IEEE Explore, June 2016

[27] Bresenham's line algorithm code, *http://rosettacode.org/wiki/Bitmap/Bresenham%27sline_algorithmPyt*

[28] Bresenham's line algorithm, *https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html*

[29] Canazza Sergio and Fantozzi Carlo and Pretto Niccolò. *Accessing Tape Music Documents on Mobile Devices.* International Journal on Digital Libraries, New York, NY, USA, 2005

[30] Micheloni Edoardo, Pretto Niccolò and Canazza Sergio *A step toward AI tools for quality control and musicological analysis of digitized analogue recordings: recognition of audio tape equalizations.* 2017

[31] Verde Sebastiano and Pretto Noccolò and Milani Simone and Canazza Sergio. *Stay True to the Sound of History: Philology, Phylogenetics and Information Engineering in Musicology.* Applied Sciences, 2018

[32] Pretto Niccolò and Fantozzi Carlo and Micheloni Edoardo and Burini Valentina and Canazza Sergio *Computing Methodologies Supporting the Preservation of Electroacoustic Music from Analog Magnetic Tape*. Computer Music Journal, 2019

[33] Mattiazzi Riccardo *Sviluppo e sperimentazione di un software per il riconoscimento automatico di discontinuità nei documenti sonori su un nastro magnetico*. Tesi di laurea Triennale, 2019