

UNIVERSITÀ POLITECNICA DELLE MARCHE

---

FACOLTÀ DI INGEGNERIA  
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E  
DELL'AUTOMAZIONE



**Automatic pain detection:  
sviluppo di un approccio basato su reti  
neurali convoluzionali per rilevare la  
presenza del dolore da espressioni del  
volto**

*Automatic pain detection: development of an approach based on  
convolutional neural networks to detect pain from facial expressions*

---

RELATORE:

**Prof. Aldo Fanco Dragoni**

CANDIDATA:

**Lorenza Gianvittorio**

CORRELATORE:

**Prof. Sernani Paolo**

ANNO ACCADEMICO 2020/2021

## *Ringraziamenti*

*Per lo svolgimento del presente lavoro in primis ringrazio il professor Aldo*

*Franco Dragoni, relatore di questa tesi di laurea, non solo per la conoscenza e l'aiuto fornitomi durante tutto l'anno, ma soprattutto per la grande disponibilità dimostratami lungo l'intero percorso di realizzazione del tirocinio.*

*Un particolare grazie va a Selene Tomassini, per la sua prontezza, repentinità, gentilezza e pazienza, per i suoi preziosi consigli non solo in ambito universitario, e per la grande complicità sviluppatasi in questo periodo, grazie per esserci stata durante tutto l'anno e per aver reso più leggero lo zaino del mio cammino.*

*Grazie inoltre anche a tutti i membri dell'AIRLab per il loro competente e fondamentale aiuto nell'implementazione del progetto.*

*Un ringraziamento speciale va ai miei genitori e alle mie sorelle, che mi hanno sempre sostenuto moralmente ed economicamente durante tutto il mio percorso, per avermi incoraggiata in ogni mia prova ed avermi dato la forza di lottare con tutta me stessa. Grazie per aver sempre creduto in me, anche quando io non ci credevo.*

*Grazie infine a tutti miei amici, per aver arricchito questo mio breve ma intenso viaggio con risate e momenti indimenticabili e per aver condiviso con me una parte di strada insieme.*

## Abstract

Il rilevamento automatico del dolore dalle espressioni facciali, in particolare dalle immagini del viso che mostrano la salute di un paziente, rimane una sfida significativa nell'area della diagnostica medica. I sistemi esperti che analizzano le immagini delle espressioni facciali, utilizzando un algoritmo di apprendimento automatico, possono essere un approccio promettente per l'analisi della presenza del dolore nel dominio della salute. Le reti neurali profonde e le tecniche emergenti di apprendimento automatico hanno compiuto progressi significativi sia nell'identificazione delle caratteristiche, che nella classificazione del dolore dalle immagini del volto, con un grande potenziale per aiutare gli operatori sanitari nella diagnosi di determinate condizioni mediche. In questa tesi, per svolgere il task di pain detection proponiamo una classificazione binaria con lo scopo di confrontare le performance fra tre diversi modelli di reti neurali pre-addestrate. Per esplorare la robustezza dell'algoritmo proposto, il database UNBC-McMaster Shoulder Pain Archive, composto da immagini del viso umano, è stato prima processato, quindi utilizzato per l'addestramento e il test dei modelli di classificazione insieme a due reti pre-addestrate deputate al task di estrazione delle features facciali: la VGG16 e la ResNet50. I risultati hanno indicato che l'algoritmo di classificazione sviluppato implementando la rete VGG16 pre addestrata su VGGFace è un potenziale strumento per il task di pain detection dalle immagini delle espressioni facciali e, pertanto, può essere adottato come strumento di intelligenza artificiale nella diagnostica medica per rilevamento automatico del dolore e successiva gestione ottimizzata dei pazienti.

# Indice

<b>Introduzione</b>	<b>6</b>
<b>1 Il dolore</b>	<b>7</b>
1.1 Aspetti attentivi e di risposta omeostatica . . . . .	9
1.2 Aspetti cognitivi, emozionali e psicopatologici del dolore . . .	9
<b>2 Le espressioni facciali e gli indicatori di dolore</b>	<b>10</b>
2.1 Le espressioni del dolore . . . . .	11
2.2 La codifica FACS e PSPI . . . . .	13
<b>3 Il riconoscimento del dolore dalle espressioni facciali con l'intelligenza artificiale</b>	<b>18</b>
3.1 Machine e deep learning . . . . .	18
3.1.1 Apprendimento automatico . . . . .	19
3.2 Transfer learning e fine tuning . . . . .	25
3.3 Stato dell'arte - review articoli . . . . .	30
3.3.1 Metodi basati sul deep learning . . . . .	30
3.3.2 Metodi non basati sul deep learning . . . . .	34
<b>4 Metodologia</b>	<b>39</b>
4.1 Dataset . . . . .	39
4.2 Pre-processing delle immagini . . . . .	46
4.3 Estrazione delle features e classificazione . . . . .	51
4.3.1 Primo modello: rete VGG-16 pre-addestrata su dataset ImageNet . . . . .	60
4.3.1.1 Pain_detection_model1_v1 . . . . .	63
4.3.1.2 Pain_detection_model1_v2 . . . . .	69
4.3.2 Secondo modello: rete VGG-16 pre-addestrata su dataset di volti . . . . .	71
4.3.2.1 Pain_detection_model2_v1 . . . . .	73
4.3.2.2 Pain_detection_model2_v2 . . . . .	77
4.3.3 Terzo modello: rete ResNet-50 pre-addestrata su dataset di volti . . . . .	78
4.3.3.1 Pain_detection_model3_v1 . . . . .	79
<b>5 Risultati</b>	<b>82</b>
5.1 Metriche di classificazione . . . . .	83
5.2 Performance del modello . . . . .	87

5.2.1	Primo modello: rete VGG-16 pre-addestrata su dataset ImageNet . . . . .	88
5.2.1.1	Pain_detection_model1_v1 . . . . .	88
5.2.1.2	Pain_detection_model1_v2 . . . . .	89
5.2.2	Secondo modello: rete VGG-16 pre-addestrata su dataset di volti . . . . .	91
5.2.2.1	Pain_detection_model2_v1 . . . . .	91
5.2.2.2	Pain_detection_model2_v2 . . . . .	92
5.2.3	Terzo modello: rete ResNeto-50 pre-addestrata su dataset di volti . . . . .	94
5.2.3.1	Pain_detection_model3_v1 . . . . .	94
<b>6</b>	<b>Discussione dei risultati e conclusione</b>	<b>95</b>

## Introduzione

Nella seguente tesi, verrà esaminato il problema del riconoscimento automatico del dolore, ovvero l'implementazione di un sistema automatico (quindi senza il diretto intervento dell'uomo) per il rilevamento di un volto sofferente, mediante l'utilizzo delle deep neural networks. Il software qui presentato e spiegato, è stato pensato per aiutare prevalentemente imprese nell'ambito sanitario, e specificatamente per tutti quei pazienti con una grave impossibilità motoria. Il rilevamento del dolore aiuterebbe dunque a ridurre i tempi di intervento nel caso il paziente risulti con un'espressione facciale dolorosa, ed ottimizzare la gestione di tutti gli altri pazienti in caso contrario. Nel presente lavoro dunque, andremo a discutere di come e quanto sia importante avere un sistema di pain detection, ed approfondiremo scendendo particolarmente nel dettaglio, tutti i passi compiuti per il suo svolgimento. I temi che andremo ad affrontare vengono riportati di seguito.

- Capitolo 1: nel primo capitolo parleremo in modo generale di cosa sia il dolore nella sua classificazione delle emozioni umane, e di come le espressioni facciali possano aiutare a riconoscere. Pertanto discuteremo di come questo particolare stato d'animo sia importante da monitorare per conservare il benessere generale.
- Capitolo 2: nel secondo capitolo parleremo della metodologia scelta per arrivare a fare detection del dolore, ovvero l'estrazione di features facciali. Discuteremo di tutti gli indicatori utilizzati e tratteremo di un sistema particolare che incorpora un metro di misura impiegato per classificare un volto sofferente da uno non sofferente. Ci soffermeremo infine su un indicatore particolare, in quanto sarà quello utilizzato nel presente lavoro.
- Capitolo 3: nel terzo capitolo riprenderemo il tema precedentemente discusso nel capitolo 2, dando una prospettiva di realizzazione mediante l'intelligenza artificiale. Introdurremo brevemente cos'è l'intelligenza artificiale e di come sia possibile completare la fase di features extraction grazie all'impiego di tecniche di machine learning. In particolare parleremo di:
  1. Caratteristiche e tratti distintivi del machine e deep learning
  2. Tecniche di transfer learning e fine tuning per i processi di feature extraction e classificazione

3. Review sullo stato dell'arte e precedenti articoli che trattano il medesimo tema.
- Capitolo 4: in questo capitolo scenderemo particolarmente nel dettaglio, discutendo dei metodi di sviluppo del software e dell'implementazione dell'algoritmo. Pertanto andremo a presentare degli script in python opportunatamente realizzati per le diverse fasi affrontate. Quindi mostreremo dapprima il database utilizzato, poi la fase di pre processing del dato ed infine la fase di sviluppo del modello per eseguire gli step di feature extraction e classificazione.
  - Capitolo 5: nel quinto capitolo discuteremo dei risultati della tecnica implementata e precedentemente esplicitata nel capitolo 4. In particolar modo mostreremo le metriche impiegate per valutare il lavoro svolto ed evidenzieremo le performance del modello realizzato.
  - Capitolo 6: nell'ultimo capitolo infine saranno trattate e discusse le considerazioni finali e le conclusioni del lavoro progettuale, tenendo presenti sia le metriche del modello, spiegate nel capitolo 5, sia lo sviluppo dell'algoritmo, illustrato nel capitolo 4.

## 1 Il dolore

Il dolore è un'esperienza sensoriale percepita a livello del sistema nervoso centrale come un'emozione sgradevole.

Con il termine nocicezione si intendono tutti quei meccanismi di trasmissione dello stimolo doloroso dalla periferia al sistema nervoso centrale; l'interazione delle diverse aree cerebrali che elaborano il segnale nocicettivo, ne consente la presa di coscienza, così si ha la percezione del dolore. Tale percezione dolorifica può essere suddivisa in diverse componenti:

- una componente sensitivo-discriminativa che consente di localizzare lo stimolo e di quantificarne qualità ed intensità;
- una componente affettivo-emozionale che consente all'individuo di reagire al dolore affettivamente parlando;
- una componente cognitivo-valutativa, che consta nella capacità di valutazione del dolore a seconda dell'educazione ricevuta e del contesto sociale.

Analizzando nel dettaglio il percorso dello stimolo doloroso dalla periferia al sistema nervoso centrale, vengono coinvolti:

- i nocicettori: sono i rivelatori dello stimolo doloroso a diversi livelli; la natura dello stimolo può essere di tipo chimico, meccanico-fisico o termico;
- lo stimolo viene poi condotto lungo le fibre nocicettive, successivamente al nervo sensitivo e dopo al nervo spinale; le fibre nocicettive possono essere amieliniche, in tal caso trasmettono lo stimolo lentamente e sono responsabili del dolore tedioso, oppure mieliniche, ovvero possono trasmettere lo stimolo più velocemente e sono responsabili del dolore puntorio;
- il nervo spinale giunge fino alla materia grigia del midollo spinale, dove a livello delle corna spinali forma due fasci: il fascio spinotalamico, che trasmette stimoli di tipo cutaneo, somatico e viscerale, risalendo il midollo fino al talamo; ed il fascio spino reticolare, che trasmette stimoli di tipo somatico e viscerale.

Giunti al sistema nervoso centrale lo stimolo viene analizzato ed integrato in diversi livelli. A livello bulbare vengono coinvolti alcuni nuclei per l'analisi dello stimolo, da qui si dipartono: vie discendenti in grado di modulare la percezione dolorosa, mediante neurotrasmettitori; altre fibre che giungono fino ai muscoli per permettere la reazione motoria allo stimolo doloroso; ed altre fibre ancora che giungono al sistema cardiovascolare e respiratorio. A livello mesencefalico si hanno le reazioni emozionali al dolore, da qui si dipartono le fibre che regolano la reazione neuroendocrina. A livello talamico (corteccia cerebrale) si ha la percezione sensitivo-discriminatoria del dolore, ovvero si localizza la provenienza dello stimolo, la qualità e quantità, cui corrisponderanno reazioni motorie e sensoriali. Tali livelli rappresentano le vie discendenti del dolore, le quali ne modulano la percezione a livello sovra spinale, nella sostanza grigia periacqueduttale e nella regione ventro-mediale del bulbo. La modulazione del dolore avviene mediante la liberazione di neurotrasmettitori, soprattutto oppioidi endogeni; la loro produzione varia da individuo ad individuo, perciò si parla di "soglia del dolore". Le vie discendenti hanno come bersagli i neuroni nocicettivi spinali e gli interneuroni (inibitori o eccitatori); gli stessi neuroni delle vie ascendenti spino-talamiche. Dopo aver analizzato dal punto di vista biologico cosa accade quando si è di fronte ad un'emozione di dolore, parliamo ora di due principali categorie di dolore e malessere.



## 1.1 Aspetti attentivi e di risposta omeostatica

*Quando ci si fa male, si percepisce immediatamente dolore ed è facile che l'attenzione venga rivolta subito verso la parte del corpo dolorante che, al contempo, si ritrae rapidamente. Tutto questo accade in maniera spontanea portando a interrompere qualsiasi attività e catalizzando tutte le risorse attenzionali verso la fonte dolorifica*

– Prince(1988)

L'attenzione è così importante nel definire il modo in cui viene percepito lo stimolo che utilizzando tecniche di distrazione, si riduce sensibilmente l'intensità dolorifica percepita e la tolleranza (per esempio è stato utilizzato efficacemente il visore di realtà virtuale). La cattura dell'attenzione non si realizza casualmente, ma è finalizzata a determinare una risposta nei confronti della fonte dello stimolo doloroso. Detto in altri termini: l'attenzione è un meccanismo di selezione per l'azione: quando si sente dolore ne consegue una spinta "arcaica" finalizzata alla fuga dalla fonte dello stimolo nocivo [6]. Questa "spinta" ovviamente è sostenuta dall'attivazione del nostro sistema omeostatico (HPA) che si attiva quando il nostro sistema rileva uno stimolo stressogeno, di cui il dolore è il miglior rappresentante.

Non è di per sé sorprendente osservare che il dolore catturi in toto la nostra attenzione e che il suo fine principale sia quello di attivare sistemi di regolazione omeostatica che hanno l'obiettivo di rispondere o fuggire alla fonte dello stimolo.

## 1.2 Aspetti cognitivi, emozionali e psicopatologici del dolore

Capita spesso che in caso di dolore persistente, ci si possa sentire nervosi, oppure depressi, come se il dolore condiziona l'intera giornata. Riprendendo la definizione proposta dalla IASP ovvero il dolore è "è un'esperienza sensoriale ed emotiva spiacevole associata a (o simile a quella associata a) un danno tissutale potenziale o in atto", come si legge, il danno tissutale può essere in atto o potenziale. Questo è un punto fondamentale perché sottolinea come la percezione del dolore sia influenzata dalla nostra interpretazione e valutazione. Quando si prova dolore, soprattutto a livello cronico, il sintomo che più spesso vi si associa è l'ansia. Paura ed ansia portano il paziente ad anticipare il dolore che proverà, esacerbando di conseguenza la sensazione. Inoltre, l'ansia anticipatoria correlata al dolore, può portare a gravi livelli di disabilità, poiché conduce all'evitamento massivo di tutte quelle situazioni e

luoghi (anche lavorativi e scolastici) dove il soggetto ha sperimentato dolore [7]. La depressione è forse il sintomo più comune, secondo la letteratura essa è presente in una percentuale che varia tra il 40% e il 50% nelle persone che soffrono di dolore cronico. Tuttavia, non sembra essere tanto la sensazione dolorifica in sé a generare lo stato depressivo, quanto le difficoltà nel farvi fronte e le ricadute sulla vita quotidiana. Ancora una volta non è la sensazione dolorifica a determinare una disfunzionalità quanto l'esperienza di dolore in generale.

Infine, insieme all'ansia e alla depressione troviamo la rabbia, che nell'individuo con sofferenza cronica di solito viene repressa [9], perché socialmente indesiderabile, questo conduce ad una maggiore probabilità di trovare soggetti che rivolgono la rabbia verso se stessi piuttosto che verso gli altri. Anche in questo caso il nostro stato emotivo (rabbia) e il giudizio della situazione sociale determinano in modo importante il modo in cui esprimiamo e processiamo l'esperienza dolorifica.

Insieme a questi fenomeni cognitivo-affettivi abbiamo costruiti “puramente” cognitivi. Tra i più importanti troviamo le credenze e la tendenza a catastrofizzare.

*Il dolore si differenzia molto chiaramente dagli altri sistemi sensoriali poiché nell'elaborazione di una percezione identificata come dolore, la sensazione, l'emozione e la cognizione sono strettamente legate*

– Le Bars e Willer, 2004 [15]

## 2 Le espressioni facciali e gli indicatori di dolore

In questo capitolo andremo a spiegare come viene riconosciuto il dolore mediante le espressioni facciali umane, vedremo come la storia può insegnarci che da sempre l'essere umano ha le stesse reazioni di fronte ad una situazione che genera dolore, e a tal scopo nella seconda parte del capitolo, parleremo delle codifiche necessarie affinché possa essere riconosciuto uno stato di sofferenza dalle espressioni del volto, ovvero i due indicatori per eccellenza:

- il sistema di codifica FACS
- l'indicatore d'intensità dolorifica PSPI

## 2.1 Le espressioni del dolore

L'evoluzione ci ha dotato di sistemi complessi per affrontare gli infortuni, molti dei quali si basano sul comportamento. I riflessi di astinenza, per esempio, esistono fin dai primi istanti e ci permettono di allontanarci dalle fonti di dolore. Gli adulti sani hanno un linguaggio, che consente loro di parlare dei dolori, delle loro cause e delle opzioni per affrontarli. Le espressioni facciali sono una fonte comportamentale di prove sul dolore.

Keefe et al [13] hanno descritto il comportamento del dolore nel modo seguente:

*... Le persone che hanno dolore possono vocalizzare la loro angoscia gemendo, piangendo o lamentandosi, o possono esibire posture del corpo o espressioni facciali legate al dolore. Questi comportamenti verbali e non verbali sono stati chiamati comportamenti di dolore perché servono a comunicare il fatto che si sta sperimentando il dolore (enfasi aggiunta)*

Questa descrizione sottolinea che i comportamenti del dolore sono comunicativi. Possiamo distinguere, tuttavia, tra comunicazione diretta e indiretta. Alcuni comportamenti che comunicano dolore servono principalmente a modificarlo (p. es. strofinare una parte irritata del corpo). Comportamenti come le espressioni facciali sembrano essere specificamente adattati per la comunicazione sociale. Probabilmente gli esseri umani sono sempre stati sensibili all'espressione facciale del dolore. Gli artisti dell'epoca classica hanno saputo catturare il senso della sofferenza attraverso le rappresentazioni del volto. Ad esempio, "Laocoonte e i suoi figli", una scultura raffigurante l'agonia di un sacerdote troiano, è notevole per la presenza di espressioni nel volto che rimandano ad un dolore agonizzante.



Figura 1: Laocoonte e i suoi figli

Charles Darwin [3] ha caratterizzato l'espressione del dolore così:

*... [nel dolore] la bocca può essere strettamente compressa, o più comunemente, le labbra sono retratte, con i denti serrati o macinati insieme ... gli occhi fissano selvaggiamente come se fossero inorriditi e attoniti.*

Quello di Darwin è stato il primo grande tentativo di porre lo studio dell'espressione del dolore su una base scientifica. Il suo approccio al dolore era coerente con le sue opinioni sulle emozioni in generale. Si è concentrato sui comportamenti espressivi come mezzo per comprendere l'origine e le funzioni degli stati motivazionali e affettivi. È stato riconosciuto il ruolo esplicito dei comportamenti di comunicazione non verbale come adattamenti funzionali. Allo stesso modo, ha enfatizzato i concetti di continuità interspecifica e ontogenica. Un altro studio interessante è stato effettuato da Chapman e Jones [2] che hanno usato il calore per valutare la "reazione al dolore". Hanno notato che:

*l'endpoint della reazione al dolore è stato prontamente rilevato osservando l'inizio della contrazione delle palpebre al canto esterno ... Solo un piccolo numero [di partecipanti] potrebbe apportare qualsiasi alterazione apprezzabile ... anche quando gli è stato chiesto di non sussultare finché possibile*

Queste osservazioni sono interessanti perché implicano che la regione facciale che circonda l'occhio sia portatrice di importanti informazioni sul dolore. Per tale motivazione, la ricerca tecnologica nell'ultimo decennio si è concentrata principalmente nel rilevamento del dolore dalle caratteristiche estratte nel volto.

A tal scopo, però, è necessario avere un sistema che sia in grado di oggettivizzare il dolore, e dare una formalizzazione specificandone una codifica universale.

## 2.2 La codifica FACS e PSPI

Le espressioni facciali sono complesse, si evolvono nel tempo e sono difficili da descrivere e quantificare. Normalmente, non lasciano traccia. Queste difficoltà rappresentano una formidabile barriera all'analisi scientifica. Lo sviluppo relativamente recente della tecnologia video a prezzi accessibili ha superato questo problema e ha stimolato la crescita del settore.

La tecnologia, tuttavia, non risolve il problema della quantificazione. Il principale progresso che ha affrontato questo problema è stato lo sviluppo di sistemi di osservazione chiaramente descritti, riducendo al minimo l'inferenza e massimizzando l'"oggettività". (La parola "oggettività" è tra virgolette perché tutti i sistemi di codifica facciale applicati dagli esseri umani implicano qualche elemento di giudizio, con la soggettività intrinseca che ciò comporta.) Il sistema di codifica dell'azione facciale (FACS - Facial Action Coding System) di Ekman e Friesen [5] è stato sicuramente il più influente. Si avvale di un principio anatomico e descrittivo: le espressioni facciali sono espresse in termini di 44 "unità di azione", che sono i cambiamenti unici prodotti dai singoli muscoli facciali o dalle combinazioni muscolari. Questo metodo ha l'intento di andare a studiare in che modo le contrazioni dei muscoli facciali, singolarmente o in combinazione con altri muscoli, modificano le sembianze di un viso. Ekman, lungo molti anni di ricerca, è riuscito a catalogare più di 10.000 combinazioni possibili di movimenti facciali, categorizzandole in 44 AU (Action Units). Ad ogni combinazione corrisponde un FACS number che comprende il nome latino dei muscoli coinvolti e l'emozione ad essi associata. La figura 2 rappresenta un insieme di AU associati ad uno specifico sentimento.

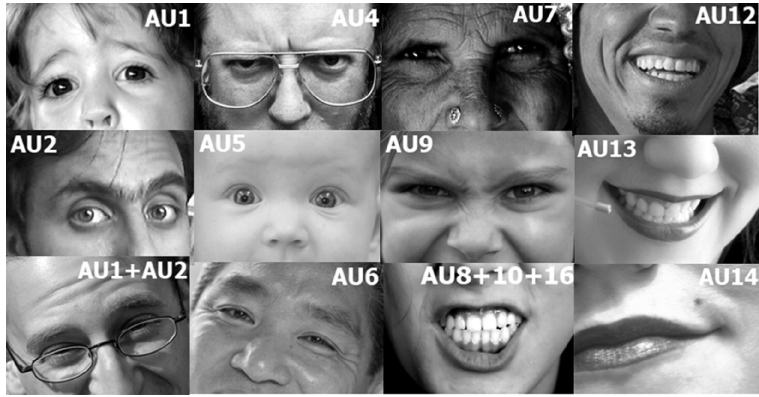


Figura 2: Esempio codifica con AU per emozioni di base corrispondenti

Un interessante particolare di questo “inventario” è che molte combinazioni di muscoli non significano nulla. Si può quindi affermare che il FACS svolge la funzione di un dizionario delle espressioni facciali che le decodifica e le associa ad emozioni specifiche.

Nello specifico, questo dizionario fornisce punteggi sulla frequenza di 7 “emozioni di base”:

- RABBIA
- TRISTEZZA
- FELICITA’
- PAURA
- DISGUSTO
- SORPRESA
- DISPREZZO
- DOLORE

Una caratteristica importante delle emozioni basilari è data dal fatto che vengono espresse universalmente, cioè da tutti, in qualsiasi luogo, tempo e cultura attraverso modalità simili. Vi sono, poi, le cosiddette “sotto-emozioni” o emozioni secondarie che invece sono specifiche di ogni contesto socio-culturale. Queste emozioni vengono tutte registrate come cambiamenti dei muscoli facciali, cioè, ad esempio, come cambiamento dei muscoli della

fronte, delle sopracciglia, delle palpebre, delle guance, del naso, delle labbra e del mento.

Una lista di tutte le possibili combinazioni di microespressioni faciali, codificate con le Action Units, può essere consultabile al seguente sito: [https://en.wikipedia.org/wiki/Facial\\_Action\\_Coding\\_System](https://en.wikipedia.org/wiki/Facial_Action_Coding_System), di cui riporteremo solo una parte della tabella proposta.

4	Brow lowerer	depressor glabellae, depressor supercillii, corrugator supercillii
5	Upper lid raiser	levator palpebrae superioris, superior tarsal muscle
6	Cheek raiser	orbicularis oculi (pars orbitalis)
7	Lid tightener	orbicularis oculi (pars palpebralis)
8	Lips toward each other	orbicularis oris
9	Nose wrinkler	levator labii superioris alaeque nasi
10	Upper lip raiser	levator labii superioris, caput infraorbitalis
11	Nasolabial deepener	zygomaticus minor
12	Lip corner puller	zygomaticus major
13	Sharp lip puller	levator anguli oris (also known as caninus)
14	Dimpler	buccinator
15	Lip corner depressor	depressor anguli oris (also known as triangularis)
16	Lower lip depressor	depressor labii inferioris
17	Chin raiser	mentalis
18	Lip pucker	incisivii labii superioris and incisivii labii inferioris
19	Tongue show	
20	Lip stretcher	risorius w/ platysma
21	Neck tightener	platysma
22	Lip funneler	orbicularis oris

Figura 3: Tabella AU e muscoli associati

Come si può osservare: a sinistra troviamo il numero identificativo dell'AU, e nelle colonne di destra viene descritto il movimento muscolare corrispondente, ed il suo nome originale in latino.

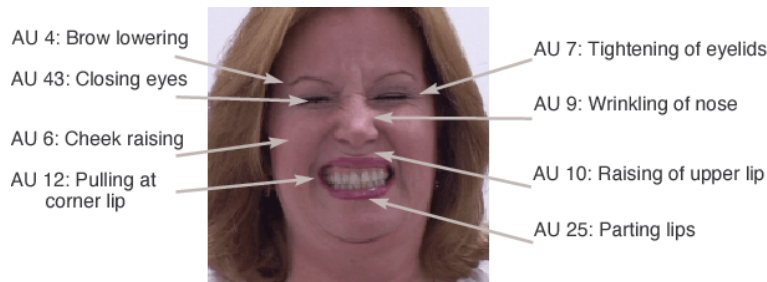


Figura 4: AU associati ad un'espressione dolorosa

La seguente immagine invece, riporta ciò che abbiamo precedentemente descritto nella tabella, applicato ad un caso visivo specifico dell'emozione intensa del dolore.

C'è da considerare anche il caso di mimica emozionale, ovvero un'emozione "menzogna", in cui l'individuo simula un'espressione, ma non prova davvero lo stato d'animo ad esso legato. Le espressioni autentiche, sentite, attivano il movimento spontaneo di alcune regioni muscolari del volto; è possibile simularle, ma in modo, in genere, poco convincente.

Quelle false, invece, sono intenzionali ed implicano l'innescio volontario di una "maschera": servono, in questo caso, a nascondere ciò che si prova veramente o a mostrare qualcosa che non si sente.

Nel libro "I volti della menzogna", Ekman classifica tre fattori per appurare che un'espressione non sia genuina e sincera:

- ASSIMMETRIA: in un'espressione facciale asimmetrica le stesse azioni si manifestano identiche nelle due metà del viso, ma sono più marcate su un lato rispetto all'altro;
- TEMPO: le espressioni che durano per più di 10 secondi probabilmente sono false;
- COLLOCAZIONE NEL DISCORSO: le espressioni del volto che non sono sincronizzate coi movimenti del corpo rappresentano possibili indizi di falsità.

Esempio: il caso del SORRISO FALSO.

E' possibile riconoscere un sorriso falso da uno genuino. Nel diciannovesimo secolo il neurologo francese Duchenne de Boulogne identificò le peculiarità del sorriso vero e genuino che coinvolge, oltre ai muscoli della bocca, anche quelli degli occhi. Infatti, nel sorriso autentico è tipica la contrazione spontanea del muscolo pars lateralis che è uno dei muscoli oculari.

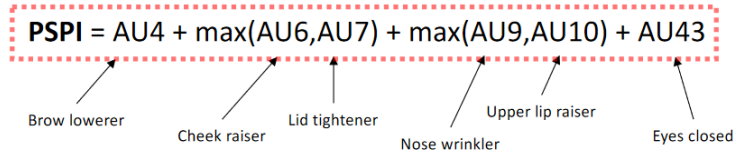
Attraverso la misurazione dell'attività cerebrale, Paul Ekman ha rilevato che nelle persone che contraggono tale muscolo è presente un'attività cerebrale legata alle sensazioni di piacere; quando si sorride in modo menzognero ciò non si verifica.

Questo grande bagaglio di scoperte ha colto l'attenzione degli Psicologi e dal 1980 il FACS cominciò ad essere utilizzato in maniera diffusa. Per tale ragione, la codifica FACS è stata di fondamentale importanza nello svolgimento del presente progetto, ne faremo dunque, un ampio utilizzo avvalendoci delle informazioni precedentemente descritte.

Una metrica molto interessante, di cui ci avvarremo, è stata appositamente utilizzata per studiare il comportamento dell'individuo e categorizzare



gli AU esclusivamente per l'emozione del dolore. Tale scala prende il nome di *Prkachin and Solomon Pain Intensity* (PSPI) che, come espresso dal nome, va a misurare diversi livelli di intensità dolorifica, basandosi sulla codifica FACS. I due studiosi Yvan Prkachin e Patricia E Solomon, si sono resi conto che nella manifestazione di uno stato di sofferenza non tutti i 44 AU erano presi in considerazione con la stessa intensità, ma solo alcuni di loro erano davvero rilevanti in quanto coinvolgevano muscoli facciali universali e tipici di un'emozione dolorosa (Vedi capitolo 2.1). In particolare gli AU coinvolti nel rilevamento del dolore sono: l'AU 4, il 6 o il 7, l'8 o il 9 ed il 43 (il gruppo muscolare corrispondente viene riportato in figura 6). Avendo questa consapevolezza ed avvalendosi della seguente formula da loro realizzata:

$$\text{PSPI} = \text{AU4} + \max(\text{AU6}, \text{AU7}) + \max(\text{AU9}, \text{AU10}) + \text{AU43}$$


The diagram illustrates the PSPI formula with arrows pointing from specific facial muscle names to the corresponding AU terms in the formula. The formula is enclosed in a red dashed box. The arrows point from the following labels to the formula:

- Brow lowerer points to AU4
- Cheek raiser points to AU6 and AU7
- Lid tightener points to AU9 and AU10
- Nose wrinkler points to AU43
- Upper lip raiser points to AU43
- Eyes closed points to AU43

Figura 5: PSPI formula

Prkachin e Solomon hanno categorizzato 16 livelli di intensità totale del dolore, associato alla somma delle singole intensità degli AU corrispondenti. In questo modo la nostra attenzione è rivolta esclusivamente ad un gruppo ristretto di AU, semplificando l'identificazione di un volto affetto da dolore rispetto ad altre tipologie di emozioni. Tale classificazione sarà ripresa ed approfondita mediante l'utilizzo di vari esempi, nel capitolo dedicato alla spiegazione della metodologia utilizzata per il seguente lavoro (capitolo 4.1).

Per concludere, abbiamo approfondito la teoria dietro alle tecniche più utilizzate per la classificazione dell'intensità di un'emozione esternata da un'espressione facciale. Nel lavoro troveremo continui riferimenti alle due codifiche principali (FACS e soprattutto PSPI) concentrandoci esclusivamente sull'emozione del dolore, in quanto, come spiegato nel presente capitolo, risultano di particolare efficacia per fare detection di un volto sofferente.

## 3 Il riconoscimento del dolore dalle espressioni facciali con l'intelligenza artificiale

### 3.1 Machine e deep learning

Sempre più spesso si sente impropriamente parlare di Intelligenza Artificiale (IA), di Machine Learning (ML) e Deep Learning (DL), come se tali termini fossero sinonimi del primo. Per una chiarezza aggiuntiva da parte del lettore, in questo capitolo andremo ad approfondire la differenza tra machine learning e deep learning, mostrando la loro correlazione con l'ambito dell'intelligenza artificiale.

L'Intelligenza Artificiale è nata come una disciplina volta a coinvolgere tutte le operazioni caratteristiche dell'intelletto umano ed eseguite da computer, quali la pianificazione, la comprensione del linguaggio, il riconoscimento di oggetti e suoni, l'apprendimento e la risoluzione dei problemi.

L'apprendimento automatico o Machine Learning è invece un modo per *attuare* l'Intelligenza Artificiale, mentre l'apprendimento approfondito o Deep Learning, è uno dei molteplici approcci relativi all'apprendimento automatico (quindi una parte a sua volta del Machine Learning).

Nella seguente figura riassumiamo la scala gerarchica delle definizioni date precedentemente.

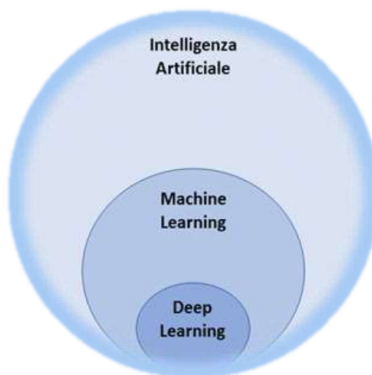


Figura 6: Gerarchia AI, ML e DL

Fatta la dovuta precisazione, scopo del presente lavoro è di approfondire le tematiche inerenti al Machine Learning e Deep Learning con i relativi campi di applicazione.

### 3.1.1 Apprendimento automatico

Il Machine Learning è un sottoinsieme dell'informatica a cui vengono applicate tecniche e modelli matematici per la realizzazione di sistemi ed algoritmi complessi, impossibili da modellare attraverso un approccio tradizionale alla programmazione.

Nel corso della vita, come esseri umani, impariamo dal mondo che ci circonda: leggendo, osservando, imparando attraverso la formazione scolastica o professionale, facendo tesoro delle esperienze, ... ecc. Da queste attività estraiamo informazioni che poi utilizzeremo nel prosieguo della nostra vita; in sostanza deriviamo delle 'regole' dalla nostra interazione con il mondo esterno. L'apprendimento è un processo iterativo, che migliora costantemente la nostra conoscenza con l'aumentare delle informazioni che raccogliamo e possiamo ben dire che più si fa esperienza e più si impara. Il Machine Learning prende spunto da tale procedimento, ovvero alla macchina viene fornito un algoritmo di apprendimento, grazie al quale essa *impara* ad attuare una specifica azione (ad esempio l'azione del riconoscimento). Una volta che la macchina ha imparato quello specifico task, è in grado di ricavare delle informazioni (modelli di conoscenza) che riutilizzerà in futuro.

Guardando il Machine Learning da una prospettiva informatica e concreta, anziché scrivere il codice di programmazione attraverso il quale, passo dopo passo, si "istruisce" la macchina su "cosa fare", al computer vengono forniti set di dati inseriti in un generico algoritmo che sviluppa una propria logica per svolgere la funzione, l'attività ed il compito richiesti.

In particolare, il Machine Learning come detto precedentemente, è una branca dell'AI che raccoglie un insieme di metodi (statistica computazionale, riconoscimento di pattern, reti neurali artificiali, filtraggio adattivo, teoria dei sistemi dinamici, elaborazione delle immagini, data mining, algoritmi adattivi, ecc.) sviluppati a partire dagli ultimi decenni del XX secolo in varie comunità scientifiche; tali metodi sono quindi utilizzati per migliorare progressivamente la performance di un algoritmo.

Il Machine Learning è impiegato principalmente per la risoluzione di tre tipologie di problemi:

- Classificazione, quando è necessario decidere a quale categoria appartiene un determinato dato;
- Regressione, ovvero prevedere il valore futuro di un dato avendo noto il suo valore attuale. Un esempio è la previsione della quotazione delle valute o delle azioni di una società. Nel marketing viene utilizzato per prevedere il tasso di risposta di una campagna sulla base di un dato

profilo di clienti; nell'ambito commerciale per stimare come varia il fatturato;

- Raggruppamento (clustering), quando si vuole raggruppare i dati che presentano caratteristiche simili. In marketing, ad esempio, il raggruppamento viene utilizzato per l'individuazione di clienti e mercati potenziali.

L'apprendimento automatico funziona in linea di principio sulla base di due distinti approcci, identificati da Arthur Samuel alla fine degli anni '50, che permettono di distinguerlo in due sottocategorie, a seconda del fatto che si forniscano al computer esempi completi da utilizzare come indicazione per eseguire il compito richiesto (apprendimento supervisionato) oppure che si lasci lavorare il software senza alcun "aiuto" (apprendimento non supervisionato).

In generale, all'interno del Machine Learning, vengono identificate le seguenti metodologie di apprendimento:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Esiste inoltre un approccio intermedio tra le prime due metodologie di apprendimento noto come Semi-Supervised Learning.

La schematizzazione delle diverse strutture utilizzate nel ML è mostrata in fig.7.

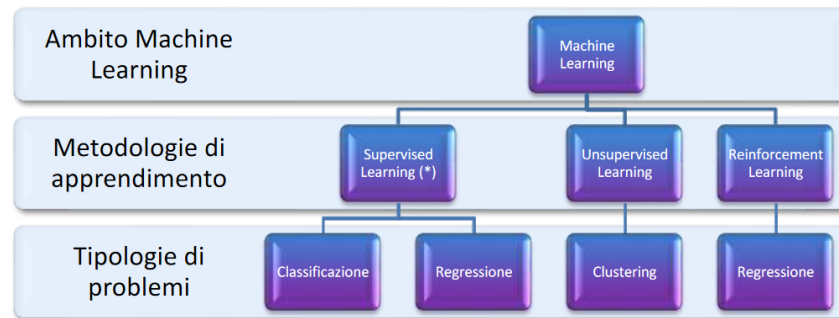


Figura 7: Metodi coinvolti nell'apprendimento automatico

Di seguito le varie metodologie di apprendimento sono descritte e messe in relazione alle tipologie di problematiche che il Machine Learning si propone di affrontare.

**L'apprendimento supervisionato:** La tecnica dell'apprendimento supervisionato consta di due fasi, training e testing; la macchina viene inizialmente istruita attraverso un set di dati (valori di input cui segue un valore di output) etichettati. Per definizione, i dati per i quali si conosce già la risposta target vengono infatti chiamati 'labeled data'. In base ai dati appresi nella fase di training, la macchina è quindi in grado di fare le opportune predizioni, con una determinata precisione da testare.

Nella tabella in fig.8 vengono riportati le categorie di applicazione, gli output e gli algoritmi specifici coinvolti per questo task.

CATEGORIA	OUTPUT	ALGORITMI
CLASSIFICAZIONE	Probabilità dell'input di appartenere ad una data categoria	<ul style="list-style-type: none"> <li>• Alberi decisionali</li> <li>• Bayesian Network</li> <li>• Support Vector Machines</li> <li>• K-Nearest Neighbor</li> <li>• Neural Network</li> </ul>
REGRESSIONE	Valore numerico ottenuto tramite combinazione lineare (somma, prodotto) di input lineari o non lineari (esponenziale, sigmoide, logaritmo...)	<ul style="list-style-type: none"> <li>• Alberi decisionali</li> <li>• K-Nearest Neighbor</li> </ul>

Figura 8: Categorie, output ed algoritmi per il Supervised Learning

In particolare si noti come classificazione e regressione si differenziano l'un dall'altro in base all'output della tecnica stessa. Nella regressione si dispone di un numero di variabili predittive (descrittive) e una variabile target continua (il risultato). In questo tipo di problema si cerca di trovare una relazione tra queste variabili al fine di prevedere un risultato.

Data una variabile predittiva  $x$  e una di risposta  $y$ , viene disegnata una retta al fine di diminuire la distanza fra i punti e la retta stessa. Il tipo di retta viene dettata dalla tecnica di regressione scelta (es. regressione lineare, non lineare, logistica ecc.).

Un esempio di regressione lineare si può notare in figura 9.

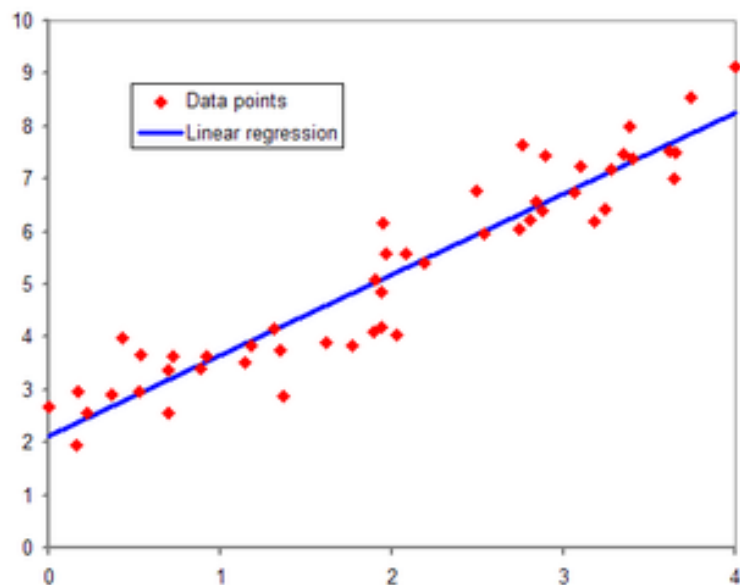


Figura 9: Esempio di approccio alla regressione lineare per predire output continui

La classificazione invece, è un metodo che si pone l'obiettivo di *categorizzare* i dati, lo scopo, in base alle analisi di dati precedentemente etichettati, è quello di riuscire a prevedere l'etichettatura delle classi di dati future. Le etichette sono valori discreti non ordinati che possono essere considerati appartenenti ad una specifica classe. La metodologia generale segue un percorso ben preciso che possiamo schematizzare ed esemplificare nella figura 10.

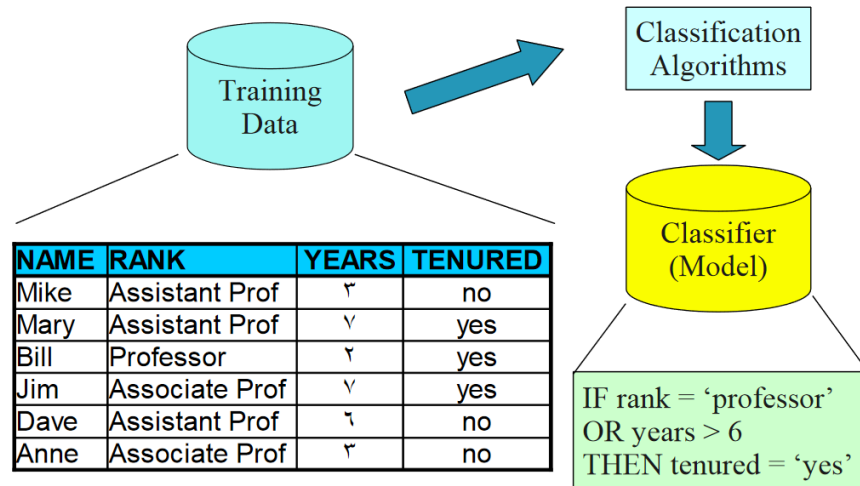


Figura 10: Schema generale seguito da un algoritmo di classificazione

Nell'esempio sopra riportato vediamo come ogni singola istanza del dataset di training (singola tupla) viene etichettata con la giusta classe di appartenenza. Il dataset viene dunque dato in pasto ad un algoritmo di classificazione che *indurrà* un modello generale, ovvero un insieme di regole in base alle quali la macchina capirà la logica di appartenenza delle singole istanze alle diverse classi.

La tecnica di classificazione, inoltre, può essere a sua volta suddivisa in due grandi categorie:

- Classificazione binaria
- Classificazione multi - classe

Nell'approccio alla classificazione binaria, l'input dell' algoritmo è un set di esempi con etichette, in cui ogni etichetta è un numero intero 0 o 1. L'output di un algoritmo di classificazione binaria è un classificatore, che può essere usato per stimare la classe di nuove istanze senza etichette. Per fare un esempio pratico consideriamo la figura 11.

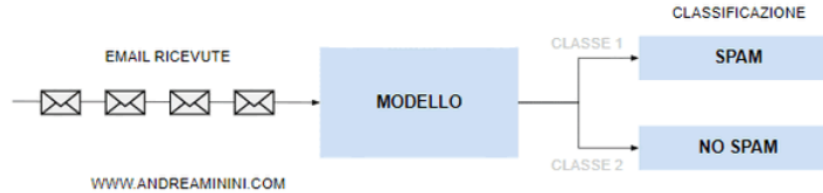


Figura 11: Esempio di approccio alla classificazione binaria

L'esempio sopra riportato, mostra la categorizzazione di email in base ad una classificazione binaria, con classi "spam" e "no spam". Seguendo la logica riportata in figura 10, l'algoritmo impara a suddividere le email e generare regole indotte, successivamente grazie ad esse può riconoscere quando un email nuova (quindi senza etichetta) risulta in una delle due classi.

La tecnica della classificazione binaria, verrà utilizzata come approccio nel riconoscimento di un volto sofferente, e verrà ampiamente spiegata nel capitolo 4.

**L'apprendimento *non* supervisionato:** mediante la tecnica dell'apprendimento non supervisionato la macchina viene istruita con un set di dati non etichettati. Lo scopo di tale strumento è raggruppare gli elementi che hanno caratteristiche simili (ovvero 'clusterizzare').

Nella tabella in figura12 vediamo esempi applicativi, con categorie, output ed algoritmi per svolgere il task di clustering.

CATEGORIA	OUTPUT	ALGORITMI
CLUSTERING	Selezione e raggruppamento di elementi omogenei in un insieme di dati: <ul style="list-style-type: none"> <li>• Clustering partizionale</li> <li>• Clustering gerarchico</li> <li>• Clustering density-based</li> </ul>	<ul style="list-style-type: none"> <li>• K-means</li> <li>• K-medoids</li> <li>• Fuzzy c-means</li> <li>• Qt clustering</li> </ul>
REGOLE DI ASSOCIAZIONE	Individuazione di associazioni interessanti e relazioni di correlazione in grandi insiemi di transazioni	<ul style="list-style-type: none"> <li>• Apriori</li> <li>• AIS</li> <li>• SETM</li> </ul>

Figura 12: Categorie, output ed algoritmi per l'unsupervised learning



Come si può leggere nella figura, il clustering consiste in un insieme di metodi per raggruppare oggetti in classi omogenee. Un cluster è un insieme di oggetti che presentano tra loro delle similarità, ma che, per contro, presentano dissimilarità con oggetti in altri cluster. L'input di un algoritmo di clustering è costituito da un campione di elementi, mentre l'output è dato da un certo numero di cluster in cui gli elementi del campione sono suddivisi in base a una misura di similarità. Gli algoritmi di clustering forniscono come output anche la descrizione delle caratteristiche di ciascun cluster, il che è fondamentale per poi prendere decisioni strategiche sulle azioni da compiere verso tali gruppi. Inoltre la misura di similarità tra un elemento e un altro può essere calcolata con metodi diversi. Esistono numerose misure di similarità, tra le quali ricordiamo la distanza di Minkowski, il Simple Matching Coefficient, il coefficiente di Jaccard, la Pearson's correlation.

Una delle misure più semplici ed utilizzate è la distanza euclidea tra due punti in uno spazio  $n$ -dimensionale.

Infine, le regole associative come vediamo dalla figura sopra riportata consistono in una serie associazioni potenziali tra diverse entità di un dataset molto grande, con l'obiettivo di individuare elementi relazionati tra loro, e valutare il piano strategico da mettere in atto. Classico esempio, che rappresenta anche gli albori delle regole associative è la *Market basket analysis*, ovvero la disciplina che analizza le abitudini di acquisto dei clienti nella vendita al dettaglio, trovando associazioni su diversi prodotti comprati, ed aiutando ad ottimizzare ad esempio, l'esposizione di diversi prodotti sugli scaffali per attirare maggiormente l'attenzione dell'acquirente.

### 3.2 Transfer learning e fine tuning

In questo capitolo affronteremo il tema dell'apprendimento trasferito, ovvero una tecnica molto utile quando si intende addestrare una rete neurale, partendo da un'altra già pre-addestrata.

Quando si parla di reti pre-addestrate, usualmente si fa riferimento a strutture neurali in cui la fase di training coinvolge migliaia (se non milioni) di samples, arrivando a concepire dei pesi, immagazzinati nella rete stessa alla fine dell'addestramento. Una rete neurale deep (profonda), è composta da un numero finito di layers, di cui i centrali vengono definiti layer nascosti (hidden layers), possiamo suddividere macroscopicamente la struttura della suddetta rete in tre parti, mostrate anche nella figura 13:

- Layer di input: è la parte che prende i dati e li manda agli strati successivi;

- Layer nascosti: processano il dato con procedimenti più o meno sofisticati, mandandolo poi in input ai successivi layers nascosti, nei quali il grado di astrazione è via via maggiore;
- Layer di output: è l'addetto alla classificazione, le uscite rappresentano le diverse classi di appartenenza.

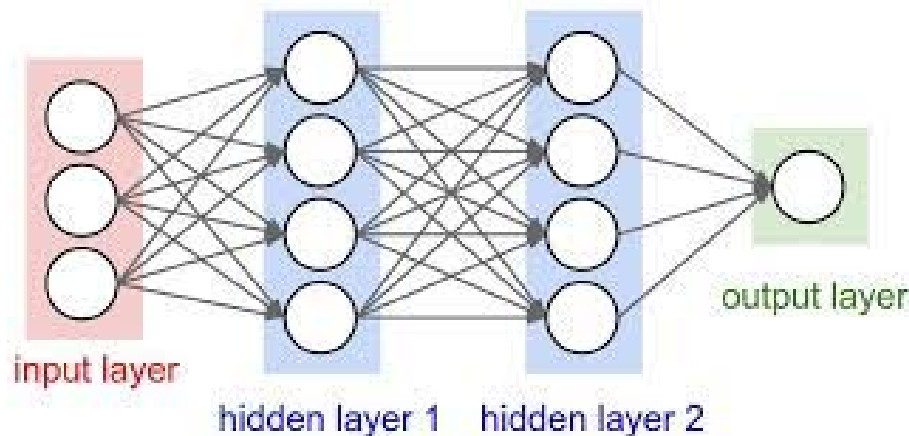


Figura 13: Esempio di rete deep con soli due layer nascosti

In una rete più complessa di quella mostrata in figura, i layer nascosti servono per processare ed estrarre caratteristiche via via sempre più complesse, più si scende in profondità, più si ottengono delle *features* sofisticate. L'idea alla base del transfer learning, è sfruttare il blocco che funge da features extractor e rimpiazzare l'ultima parte, addetta alla classificazione nella rete pre-addestrata, con un nuovo blocco incaricato di classificare le nuove categorie di interesse.

Nel procedimento inoltre, si utilizza una rete pre-addestrata su campioni correlati con il nostro dataset (ad esempio, una rete addestrata sul riconoscimento di macchine, può essere utilizzata per il riconoscimento di camion).

L'approccio al metodo dell'apprendimento trasferito, può suddividersi in due grandi categorie:

- Fine-tuning: addestrare la rete sostituendo gli ultimi layers
- Features extraction: usare solo il blocco che funge da estrattore di features

**Features extraction** La prima tecnica per l'applicazione del transfer learning è utilizzarla solo per estrarre features; quindi impiegarla esclusivamente per ottenere l'output della parte più interna della rete, semplicemente eliminando gli ultimi strati addetti alla classificazione. Come mostra la figura 14,

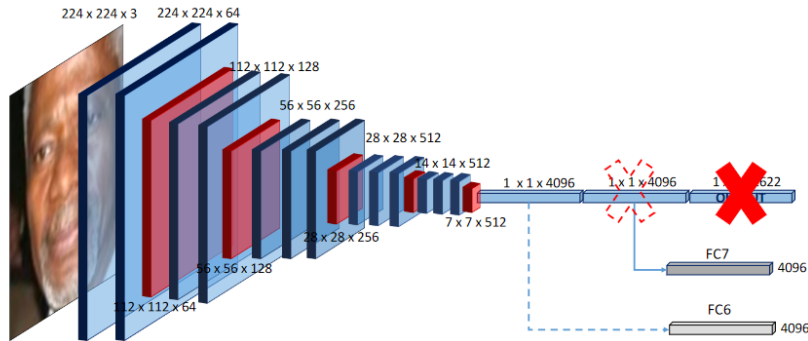


Figura 14: Esempio di approccio al transfer learning eliminando solo uno strato in testa alla rete pre-addestrata

in cui si prende in considerazione una rete pre-addestrata chiamata la VGG16, di cui parleremo in dettaglio nel capitolo 4. In questo modello composto da 16 livelli, è stato eliminato l'ultimo layer (un fully connected), addetto alla classificazione di 1000 categorie, per estrarre features dalle restanti parti, ottenendo così un feature vector composto da 4096 elementi. Il descrittore profondo così ottenuto può essere impiegato in vari e sofisticati modi, inoltre il vettore sarà più complesso a mano a mano che scendiamo in profondità della rete: se una rete è addestrata a riconoscere volti, come in esempio, il feature vector ottenuto dagli ultimi strati della rete, rappresenterà concetti come la bocca, il naso, o il volto; mentre rimanendo sulla superficie della rete stessa, otteniamo concetti molto più astratti come una linea, un triangolo, o un cerchio.

**Fine tuning:** L'utilizzo della seconda tecnica, ovvero del fine-tuning, ci consente di evitare di intaccare i pesi della prima parte della rete (chiamata anche backbone) attraverso una operazione di freezing dei layer coinvolti allo scopo di non modificare il comportamento della rete iniziale e consentirci di addestrare solamente il pezzo della rete di nostro interesse.

Il backbone, dunque, è in sostanza l'insieme di strati che vengono estratti da una rete pre-addestrata e che saranno inseriti in testa alla rete neurale che progetteremo, con lo scopo di creare un modulo specializzato all'estrazione delle features.

Il transfer learning mediante fine-tuning è molto importante in quanto è alquanto evidente che partendo da una rete neurale già pre-addestrata l'operazione di training può convergere più velocemente e può consentire alla rete neurale di avere una risposta migliore rispetto ad una addestrata from scratch con un dataset più piccolo.

In definitiva i passi coinvolti nell'applicazione di tale tecnica sono i seguenti:

- Scelta della rete pre-addestrata correlata, da cui attingere per prelevare il backbone. Una possibile lista delle reti addestrate su riconoscimento di immagini, in Keras<sup>1</sup> è consultabile a questo url: <https://keras.rstudio.com/articles/applications.html>;
- Decidere quali e quanti strati degli hidden layers “congelare” per utilizzare la backbone come features extractor nella nuova rete;
- Eliminare i restanti strati, ovvero gli addetti alla classificazione;
- Sostituire gli strati appena eliminati con dei layer di classificazione nella nuova rete.

Nella figura di seguito (fig. 15) vediamo un esempio di applicazione del fine-tuning.

---

<sup>1</sup> Keras è una libreria open source (con licenza MIT) scritta in Python, frutto principalmente del lavoro dello sviluppatore di Google François Chollet nell'ambito del progetto ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).

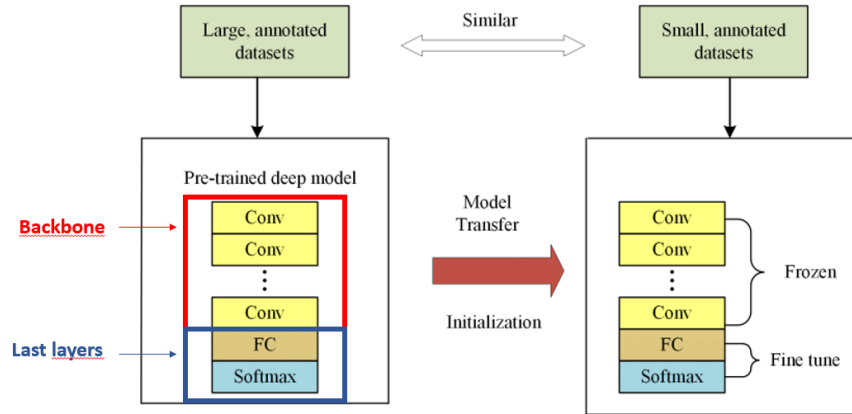


Figura 15: Esempio di approccio alla tecnica del fine-tuning

A sinistra della figura viene riportato il modello pre-addestrato con una una grande mole di dati annotati, nel blocco inferiore di tale modello si nota la presenza della backbone (in rosso), necessaria per completare l'operazione di estrazione delle features; inoltre nel modello pre-addestrato sono presenti altri due strati dopo la backbone, utilizzati per la classificazione del modello originale (evidenziati in blu). Sulla destra, sempre della stessa figura, si nota invece il nuovo modello di rete che “prende in prestito” la backbone dal modello precedente e a cui vengono passati dati diversi (tendenzialmente un database più piccolo); la backbone viene congelata con lo scopo di mantenere i pesi della rete intatti, e gli ultimi due strati vengono sostituiti con livelli da addestrare *ex novo*, addetti alla nuova classificazione di interesse.

L'applicazione del transfer learning, dunque, ci consente di ottenere due risultati:

- riutilizzo di una rete già addestrata e pronta per estrarre efficacemente feature dai dati di input
- limitazione dell'elaborazione ad un numero sensibilmente minore di parametri (corrispondenti agli ultimi layer)

Nel proseguio del lavoro tratteremo specificatamente la tecnica del transfer learning utilizzando il fine tuning, per implementare il task del riconoscimento del dolore.

### 3.3 Stato dell'arte - review articoli

In questo capitolo andremo ad approfondire le architetture presenti sul tema del riconoscimento automatico del dolore. Nelle seguenti sezioni verranno spiegati nel dettaglio tutti gli articoli reperibili pubblicamente, dettagliando per ogni paper le differenti architetture utilizzate. Per una maggior chiarezza, gli articoli verranno classificati in base alla metodologia adottata (in particolare metodi basati sul deep learning e non) e la spiegazione si concentrerà principalmente su:

- Database adottato
- Pre processing
- Metodologia di features extraction
- Metodo di classificazione
- Metriche e prestazioni

#### 3.3.1 Metodi basati sul deep learning

Nella presente sezione, analizzeremo gli articoli che utilizzano metodi basati sul deep learning.

##### **Enhanced Deep Learning Algorithm Development to detect pain intensity from facial expression images :**

L'articolo in questione è stato pubblicato a febbraio del 2020, e propone una classificazione in base all'intensità del dolore provato dal paziente, mediante supporto di features facciali. In particolare non essendo una classificazione binaria, possiede quattro livelli graduali di intensità. Il database utilizzato dal presente paper è l'UNBC-McMaster Shoulder Pain Achieve database, un archivio contenente 200 sequenze video, e 48 398 frames. Il database inoltre porpone per ogni frame, una particolare codifica dei movimenti facciali: la codifica FACS (Facial Action Coding System). Il FACS è un sistema per classificare i movimenti del volto, e si usa attribuendo una combinazione di codici corrispondenti a determinati micromovimenti facciali (chiamati AU). Sfruttando l'appena descritto database, l'articolo affronta successivamente una parte di pre processing delle immagini, soffermandosi su tre step:

- Resizing

- Cropping / Centralizing
- Normalizing

Una volta acquisite le immagini preprocessate, ogni singolo frame è passato come input al blocco di features extracting. Per estrarre features facciali viene utilizzata una rete pre addestrata di tipo convoluzionale (VGG - face Convolutional Neural Network) , sfruttando la tecnica del transfer learning. All'interno dello stesso blocco viene prevista aggiuntivamente una fase di diminuzione dimensionale delle features estratte, mediante il metodo PCA (Principal Component Analysis). A questo punto l'output viene trasferito al blocco di classificazione, che utilizza un classificatore ibrido composto da una rete CNN ed una Bidirectional Long Short Term Memory (BiLSTM).

Il metodo utilizzato, raggiunge delle prestazioni avanzate, con un'accuratezza del 91% ed un AUC (Area Under the Curve) del 98.4%.

#### **Deep Pain: Exploiting Long Short-Term Memory Networks for Facial Expression Classification :**

Il presente paper pubblicato il 9 febbraio 2017 ripropone gli stessi step dell'articolo precedente, sfruttando una metodologia simile. In particolare il database utilizzato è l'UNBC-McMaster Shoulder Pain Achieve database, ed i passi di preprocessing per ogni immagine si differenziano in:

- Cropping
- Frontalize
- Normalization

Specificatamente, la fase di frontalizing è stata ottenuta usando il metodo GAP (Generalized Procrustes Analysis), successivamente le immagini preprocessate sono state passate al blocco feature extraction, utilizzando una rete convoluzionale pre addestrata (VGG -face CNN). Questa architettura è appositamente progettata per acquisire informazioni 2D, quindi funziona molto bene sulle immagini. Tuttavia nel database sono presenti dei video, quindi si possono impiegare informazioni temporali per migliorare le prestazioni del sistema. Per raggiungere questo scopo, il paper propone di utilizzare in fase di classificazione una rete RNN - LSTM(Long Short Term Memory). Mediante questo approccio, l'articolo raggiunge un accuratezza del 90% ed un AUC del 91%.

### **Pain Detection from Facial Videos Using Two-Stage Deep Learning :**

Il metodo proposto dal seguente articolo è un approccio basato sulla detection di AU riguardanti il dolore. Precedentemente abbiamo accennato all'importanza dei micro movimenti facciali, estratti mediante il sistema di codifica FACS. Dunque Il sistema proposto è un modello di apprendimento a due fasi composto da due reti neurali profonde (DNN) collegate sequenzialmente. Il primo DNN è un modulo di rilevamento del viso che viene utilizzato per estrarre i landmark facciali in modo robusto. Il secondo DNN è una rete CNN utilizzata per il rilevamento di AU.

Procedendo per step, in questo paper troviamo tre tipologie di database utilizzati:

- Extended Cohn-Kanade Dataset
- UNBC-McMaster Shoulder Pain Achieve database
- Wilkie's dataset

La fase di pre processing dell'algoritmo proposto, ha previsto l'utilizzo del metodo Face Alignment Network (FAN), basato sul DL, che è in grado di gestire il problema dell'allineamento del viso nei set di dati 2D e 3D. La rete FAN viene utilizzata in questo studio per tracciare 68 punti fiduciali da un'immagine contenente un volto, che comprende rilevamento del volto e detection di landmark facciali. La prima parte è implementata utilizzando Single-Shot Scale-Invariant Face Detector, un rilevatore di volti in tempo reale basato su rete CNN, che è in grado di rilevare i volti a scale differenti. Per la fase di classificazione il paper propone una struttura di rete CNN pre addestrata tramite la VGG- face2, per il riconoscimento di AU. L'addestramento è stato effettuato mediante una combinazione dei primi due dataset, ed è stato infine valutato con il Wilkie's dataset. Le performance del sistema sono state illustrate utilizzando le seguenti metriche:

- F1 - score: pari all' 82%
- Accuracy: pari al 98%.

### **Spatio-temporal Pain Recognition in CNN-Based Super-Resolved Facial Images :**

In questo articolo viene proposta una classificazione in base all'intensità del dolore provato dall'individuo. Il database utilizzato anche in questo caso



è l'UNBC-McMaster Shoulder Pain Achieve database. Nel presente paper però si ha lo scopo di ottenere diversi database da testare, dunque nella prima fase è stato applicato ai dati originali una variazione di risoluzione tramite l'algoritmo SR (Super Resolution). Per generare immagini SR è stata utilizzata una tecnica chiamata example - based learning. In fase di preprocessing, per estrarre parti del viso discriminanti è stata utilizzata la tecnica del transfer learning tramite il modello pre addestrato VGG - face CNN. Una volta estratte le features, per la fase di classificazione è stata utilizzata una rete LSTM. La classificazione è avvenuta in base a tre livelli di dolore percepito:

- Strong Pain
- Weak Pain
- No Pain

ed ha portato alle seguenti metriche prestazionali:

- Accuray: 61.9%
- F1 - score: 69%

Nell'articolo inoltre viene approfondita la divergenza a livello prestazionale, tra l'utilizzo di una CNN in fase di classificazione, ed un LSTM, evidenziando gli scarsi risultati ottenuti con la prima rete.

### **Recurrent Convolutional Neural Network Regression for Continuous Pain Intensity Estimation :**

Il metodo presentato dal seguente articolo, si differenzia dai precedenti in quanto la classificazione è stata svolta in continuo. Quindi non si parla più propriamente di classi, ma di valori continui. Anche qui il database utilizzato per lo studio è l'UNBC-McMaster Shoulder Pain Achieve database. Nella prima fase di preprocessing ogni frame entrante della sequenza video è stato allineato e deformato nella stessa posa frontale, per raggiungere questo scopo è stata utilizzata la tecnica dell'AAM tracking. Infine per la fase di classificazione è stata impiegata una rete RCNN (Recurrent Convolutional Neural Network), ottenendo una classificazione a valori continui. Le prestazioni del presente metodo di pain detection sono state valutate con due metriche:

- MSE (Mean Square Error) pari a 1.54;
- PCC (Pearson Correlation Coefficient) pari a 0.65.

### 3.3.2 Metodi non basati sul deep learning

In questa sezione invece andremo ad approfondire articoli reperibili pubblicamente, che esplorano studi inerenti alla pain detection/recognition, non basati sul deep learning.

#### **Real-time pain detection in facial expressions for health robotics**

Nell'articolo qui presentato, viene implementato un metodo sofisticato sia per allenare, sia per testare la rete stessa. Il training infatti viene effettuato non solo prendendo in considerazione un dataset contenente volti sofferenti, ma anche un database con emozioni generali, per confrontare i vari risultati ottenuti. dunque i dataset utilizzati sono:

- AffectNet Dataset
- BioVid Heat Pain Dataset
- Facial Expression Recognition 2013 Dataset (FER-2013)
- UNBC-McMaster Shoulder Pain Dataset

Il primo passo nel presente lavoro è stato usare un toolkit per il riconoscimento automatico di Action Units, chiamato OpenFace. L'applicazione del suddetto toolkit, è stata utilizzata su tutti i frame nel set di dati. L'output ottenuto è stato un file CSV con informazioni per ogni frame, come il suo ID, il successo del rilevamento del volto e i valori per ogni AU. Prima di ulteriori elaborazioni, i fotogrammi in cui non è stato rilevato alcun volto da OpenFace sono stati cancellati. Successivamente sono state estratte le label per ogni frames e si è passati alla fase di allenamento.

Per fare training l'articolo ha utilizzato una SVM (Support Vector Machine), l'input alla SVM sono stati 6 valori di intensità di AU con un'etichetta di 1 o 0 (dolore/non dolore).

I risultati sono stati testati su differenti combinazioni dei dataset precedentemente elencati, raggiungendo un accuratezza dell'88%, ed utilizzando UNBC-McMaster Shoulder Pain Dataset unito all'AffectNet Dataset.

#### **Pain Recognition and Intensity Classification Using Facial Expressions**

Questo documento presenta un'analisi comparativa degli approcci basati sui filtri energetici Gabor combinati con potenti classificatori, come Support Vector Machines (SVMs), per il rilevamento del dolore e la classificazione in tre livelli.

L'intensità del dolore viene etichettata utilizzando la scala Prkachin e Solomon Pain Intensity. In questo lavoro, i livelli di intensità sono stati quantizzati in tre gruppi disgiunti:

- nessun dolore
- dolore debole
- dolore forte

Anche qui il dataset coinvolto è stato L'UNBC-McMaster Shoulder Pain Achieve database, per la fase di preprocessing è stato utilizzato un approccio comune per l'identificazione del volto, noto come l'algoritmo di Viola-Jones. Mentre per la fase di features extraction è stato impiegato il metodo del Gabor energy filter, con 8 orientamenti e 5 frequenze. Infine la fase di training ha coinvolto una SVM, utilizzando la tecnica della cross validation.

I tassi di precisione sono del 74%, 30% e 78% quando si distingue il dolore in nessun dolore, dolore debole e dolore forte rispettivamente; quindi in media viene raggiunta una precisione del 60%.

**Painful data: The UNBC McMaster shoulder pain expression archive database** L'articolo in questione risulta come il primo articolo ad introdurre il database realizzato dall'università McMaster e dall'università della Northern British Columbia, ovvero quello utilizzato dalla maggior parte degli articoli precedentemente descritti: L'UNBC-McMaster Shoulder Pain Achieve database.

Nel lavoro viene utilizzato un sistema basato su Active Appearance Model (AAM) che utilizza AAM per tracciare il viso ed estrarre le sue caratteristiche. Nella distribuzione dei dati vengono inclusi i 66 punti di riferimento AAM per ogni immagine. Successivamente all'estrazione delle caratteristiche visive sono state utilizzate queste informazioni per ricavare due tipi di features:

- The similarity normalized shape or points (SPTS)
- The canonical normalized appearance (CAPP)

I risultati sono stati i seguenti:

- 91.81% di accuracy con l'utilizzo della features combinate (SPTS+CAPP)
- 83.92% di AUC con l'utilizzo delle features combinate(SPTS+CAPP)

**Pain Recognition Using Artificial Neural Network** In questo penultimo articolo viene utilizzato un rilevatore di volti automatico ed un tracker di caratteristiche facciali rispettivamente per il rilevamento dei volti e l'estrazione delle caratteristiche. Il rilevatore facciale utilizza l'approccio di modellazione del colore della pelle. Per il riconoscimento del dolore, vengono calcolate le caratteristiche di posizione e forma dei volti rilevati. Queste caratteristiche vengono quindi utilizzate come input per la rete neurale artificiale che utilizza l'algoritmo standard error back-propagation, per la classificazione dei volti sofferenti e non.

Il dataset utilizzato essendo l'articolo risalente all'anno 2006, è stato il Psychophysiology Laboratory Database of University of Northern British Columbia (UNBC). La fase di preprocessing ha riguardato esclusivamente l'estrazione del volto umano, utilizzando una tecnica basilare e tagliando parte delle immagini, ottenendo le facce. Successivamente per la parte di features extraction sono state estratte le caratteristiche della posizione: come Centri oculari e punti finali interni del sopracciglio e gli angoli della bocca, ed infine è stata compiuta la rappresentazione delle caratteristiche della posizione, ritagliando le immagini ed estraendo le *shape features*.

Per la fase di addestramento è stata utilizzata una rete neurale artificiale (una NN) con 10 hidden layer, raggiungendo un accuracy media complessiva del 91%.

**Fusing Deep Learned and Hand-Crafted Features of Appearance, Shape, and Dynamics for Automatic Pain Estimation** In questo lavoro vengono combinate caratteristiche artigianali (hand-crafted) con caratteristiche apprese per la stima dell'intensità del dolore. Innanzitutto, vengono estratte le caratteristiche hand-crafted in base ai 66 punti di riferimento facciali. Successivamente vengono estratte le caratteristiche deep-learned dalle CNN di riconoscimento AU. Infine, viene utilizzato un modello di regressione sulle caratteristiche individuali e combinate.

Il database utilizzato anche qui è L'UNBC-McMaster Shoulder Pain Achieve database, nella prima fase di preprocessing viene estratto il volto dall'immagine utilizzando i punti di riferimento facciali forniti con il database (Landmarks). Quindi la faccia estratta viene allineata a una forma media basata su una Procrustes transformation dei punti facciali dell'occhio e degli angoli della bocca.

Per la parte di features extraction invece sono state utilizzate due CNN pre-addestrate per AU detection, le sequenze di input però sono state prima normalizzate sottraendole dall'immagine media della rete pre-addestrata,

successivamente è stato recuperato l'output dell'ultimo livello di convoluzione completamente connesso, ed infine, le caratteristiche delle due CNN (cioè le CNN della regione degli occhi e della bocca) sono state combinate per dare un ampio features set. Il tutto è stato passato ad un Relevance Vector Regressor (RVR), per completare la fase di regressione (quindi valori continui). I risultati così ottenuti sono i seguenti:

- RMSE: 0.99
- PCC (Correlation): 0.67

A conclusione di questo capitolo, nelle tabelle 16, 17 e 18 sotto riportate, viene mostrata una schematizzazione di ciò che abbiamo appena descritto, mostrando: il database utilizzato da ciascun articolo, le fasi affrontate, la data di pubblicazione, le immagini contenute del rispettivo database con la loro risoluzione, e le performance del modello utilizzato.

Figura 16: Tabella 1 - articoli presenti nello stato dell'arte

	Database	Pre - processing	Features Extraction	Classificati on	N. citazi oni	Data pubblicazio ne	N. immagini	Risoluzio ne immagini	Scelta della codifica	Metriche e prestazioni
<i>Enhanced Deep Learning Algorithm Development to detect pain intensity from facial expression images</i>	UNBC-McMaster Shoulder Pain Achieve database	1. Resizing 2. Cropping/Ce ntralizing 3. Normalizing	<ul style="list-style-type: none"> <li>• VGG – face CNN</li> <li>• PCA (riduzio ne dimensionalità )</li> </ul>	CNN + BiLSTM	18	10 febbraio 2020	200 video da 25 pazienti - (48,398 frames)	224 x 224 x 3	PSPI	ACCURACY: 91 % AUC: 98,42 % (con il CNN entrambi meno del 55 %)
<i>Deep Pain: Exploiting Long Short-Term Memory Networks for Facial Expression Classification</i>	UNBC-McMaster Shoulder Pain Achieve database	<ul style="list-style-type: none"> <li>- Cropping</li> <li>- Frontalize</li> <li>- Normalizazi on</li> </ul>	VGG – Face CNN	LSTM	165	9 febbraio 2017	48 398 frames	224 x 224 x 3	AU	ACCURACY: 90% AUC: 91 %
<i>Pain Detection from Facial Videos Using Two-StageDeep Learning</i>	1. Extended Cohn-Kanade Dataset 2. UNBC-McMaster Shoulder Pain Achieve database 3. Wilkie's dataset (per testig)	Face detection and alignment	VGG- face CNN		3	1° gennaio 2019	48 398 immagini (UNBC) + 593 video (Cohn-Kanade)	160 x 160 x 3	AU	ACCURACY: 98% F1 – SCORE: 82%

Figura 17: Tabella 2 - articoli presenti nello stato dell'arte

<b>Spatio-temporal Pain Recognitionin CNN-Based Super-Resolved Facial Images</b>	UNBC-McMaster Shoulder Pain Achieve database	Aumento di risoluzione (Super resolution technique) Per divisione in diversi database.	VGG – face CNN	LSTM	40	4 dicembre 2016	48 398 frames	224 x 224 x 3	PSPI	ACCURACY: 61.9%
<b>Real-time pain detection in facial expressions forhealth robotics</b>	1. UNBC-McMaster Shoulder Pain Achieve database 2. AffectNet 3. FER 2013 4. BioVid Heat Pain	//	OpenFace 2.0 (libreria che fa features extraction in real time) //	SVM  CNN (meno utilizzato, poiché meno accurato)	2	Dicembre 2019	48 398 ( 35 887 (FER) 24 K immagini (AffectNet) 20 video per 85 pazienti (BioVid, solo per testing)	//	AU	ACC 1: 85 % (UNBC) ACC 2: 88 % (AffectNet + UNBC)  ACC 3: 97 % (FER + UNBC) problema dell'adattamento
<b>Pain Recognition and Intensity Classification UsingFacial Expressions</b>	UNBC-McMaster Shoulder Pain Achieve database	Normalization → Viola Jones + eye normalization	Gabor energy filter (orientamento)	SVM	8	Luglio 2016	48 398 frames	96 x 106	PSPI	PRECISION: 60.6%
<b>Painful data: The UNBC-McMaster shoulder pain expression archive database</b>	UNBC-McMaster Shoulder Pain Achieve database	AAM per estrarre il volto ed estrarre features (normalized shape, canonical normalized appearance)		SVM	475	25 marzo 2011	48 398 frames	//	AU	ACCURACY: 91,8% AUC: 83.9 %

Figura 18: Tabella 3 - articoli presenti nello stato dell'arte

<b>Pain Recognition Using Artificial Neural Network</b>	Psychophysiology Laboratory Database of University of Northern British Columbia (UNBC)	- Segmentati on and face location - Face detection	Iterative thresholding method (estrarre angoli bocca, location occhi, sopracciglia ecc....)	NN (3 layers)	52	Settembre 2006	2 video per 38 pazienti	90 x 90	AU	ACCURACY: 91 %
<b>Recurrent Convolutional Neural Network Regression for Continuous Pain Intensity Estimation</b>	UNBC-McMaster Shoulder Pain Achieve database	AAM (estrazione forma del viso) + Flattering		RCNN (in continuo)	100	Luglio 2016	48 398 frames	713 x 30 x 3	AU	MSE: 1.54 PCC : 0.65
<b>Fusing Deep Learned and Hand-Crafted Features of Appearance, Shape, and Dynamics for Automatic Pain Estimation</b>	UNBC-McMaster Shoulder Pain Achieve database	- Face detection - Alignment	AU detection (CNN pre trained) + Geometric shape and location	Relevance Vector Regressor (RVR, in continuo)	43	Gennaio 2017	48 398 frames	//	AU	RMSE: 0.99 PCC ( correlation): 0.67

## 4 Metodologia

In questo capitolo verrà discussa la metodologia di sviluppo del presente lavoro, in particolare verranno trattati i seguenti temi:

- Dataset: capitolo che spiega dettagliatamente che tipo di dati sono stati utilizzati per implementare il progetto;
- Pre - processing: in questo capitolo ci occuperemo di spiegare che tecniche sono state utilizzate per la pulizia del dato, inserendo anche esempi di codice per una maggiore chiarezza;
- Estrazione delle features e classificazione: nell'ultimo capitolo invece, parleremo approfonditamente delle reti utilizzate, discutendo le performance di quattro modelli testati.

Seguiremo dunque una pipeline propedeutica per il corretto sviluppo progettuale.

### 4.1 Dataset

Il dataset scelto per il riconoscimento di volti affetti da dolore, è stato l'UNBC-McMaster shoulder pain expression archive database.

Infatti, in base alla tabella riportata nel capitolo 4 dello stato dell'arte, si nota come il suddetto database sia sicuramente il più utilizzato; questa valutazione è stata importante per guidarci nella scelta del set di dati.

L'UNBC-McMaster shoulder pain expression archive database, nasce da un'idea dei ricercatori della McMaster University e della University of Northern British Columbia che hanno catturato video dei volti dei partecipanti (affetti da dolore alla spalla) mentre eseguivano una serie di test attivi e passivi sull'ampiezza di movimento degli arti colpiti e non in due distinte occasioni.

Un totale di 129 partecipanti (63 maschi, 66 femmine) che sono stati identificati come aventi un problema con il dolore alla spalla sono stati reclutati da 3 cliniche di fisioterapia e da annunci pubblicati nel campus della McMaster University. Un quarto erano studenti e altri provenivano dalla comunità e comprendevano un'ampia varietà di occupazioni. La diagnosi del dolore alla spalla variava, con i partecipanti che soffrivano di artrite, borsite, tendinite, sublussazione, lesioni della cuffia dei rotatori, sindromi da impingement, sperone osseo, capsulite e lussazione. Oltre la metà dei partecipanti ha riferito l'uso di farmaci per il loro dolore. Tutti i partecipanti sono stati testati in una stanza di laboratorio che incluso un letto per l'esecuzione di

test di mobilità passiva. Dopo che il consenso informato e le procedure informative sono state completate, i partecipanti sono stati sottoposti a otto test standard di mobilità: abduzione, flessione e rotazione interna ed esterna di ciascun braccio separatamente. Durante i test attivi e passivi, due fotocamere digitali Sony hanno registrato le espressioni facciali dei partecipanti. L'orientamento della fotocamera era inizialmente frontale in condizione attiva, sebbene il cambiamento di posa fosse comune. Era inoltre disponibile una scheda che elencava i descrittori verbali del dolore per aiutare i partecipanti a fornire valutazioni verbali del dolore prodotto in ciascun test. Ogni scheda mostrava due scale di tipo Likert<sup>2</sup>. Uno consisteva in parole che riflettevano l'intensità sensoriale di dolore. L'altro consisteva in parole che riflettevano l'affettività di dimensione motivazionale. Queste scale sono state soggette ad approfondite analisi psicofisiche, che hanno stabilito le loro proprietà come misure di scala dei rapporti delle rispettive dimensioni sottostanti. Ogni scala aveva 15 elementi etichettati da "A" a "O".

La scala sensoriale (SEN) iniziava da "estremamente debole" e terminava con "estremamente intensa"; la scala affettivo-motivazionale (AFF) iniziava da "sopportabile" e terminava a "straziante". Inoltre, i partecipanti hanno completato una serie di scale analogiche visive (VAS) di 10 cm, ancorate a ciascuna estremità con le parole "Nessun dolore" e "Il dolore più forte possibile".

Di seguito si riportano due esempi: il primo mostrato in figura 19, spiega il metodo della scala Likert, il secondo in figura 20 mostra invece la scala analogica visiva, ed il suo funzionamento.

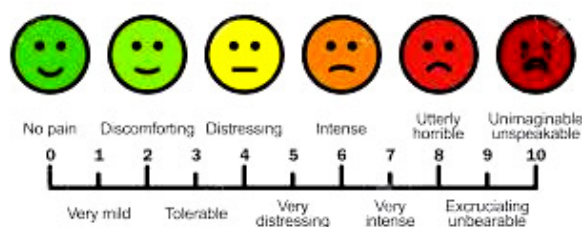


Figura 19: Esempio di scala Likert in pronto soccorso

In figura 19 invece, viene mostrata una scala presente nel pronto soccorso, che permette di avere un metro di misura universale sul dolore che

<sup>2</sup> likert



prova un determinato paziente. Come precedentemente descritto, la scala SEN e la scala AFF, sono due scale likert: la scala Likert fu ideata dallo psicometrico americano Rensis Likert nel 1932 con lo scopo di elaborare un nuovo strumento, più semplice rispetto ad altri, per la misurazione di opinioni e atteggiamenti. Essa è una scala di valutazione, in cui si presenta al candidato un'affermazione, ed il candidato deve esprimere una valutazione in base alle opzioni che la scala comprende.

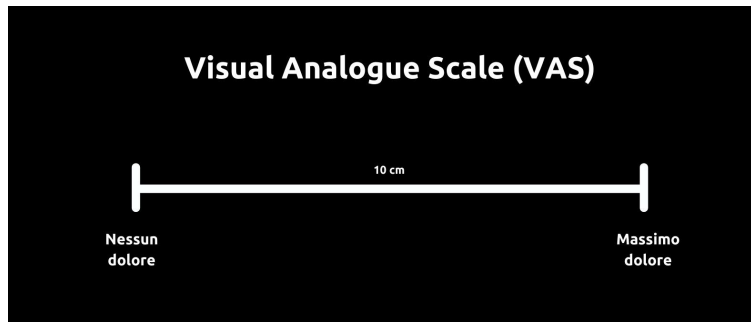


Figura 20: Esempio di VAS

In quest'altra figura invece, vi è una rappresentazione della Visual Analogue scale: Essa corrisponde alla rappresentazione visiva dell'ampiezza del dolore avvertito dal paziente ed è costituita da una linea predeterminata lunga 10 cm, dove l'estremità sinistra corrisponde a "nessun dolore", mentre l'estremità destra a "peggior dolore possibile". Al paziente viene chiesto di tracciare sulla linea un segno che rappresenti il livello di dolore provato. La linea della scala può essere orientata orizzontalmente o verticalmente, senza che questo influenzi la sua sensibilità.

Il punteggio viene calcolato in mm, misurando tramite un righello la lunghezza della linea compresa tra l'estremo che corrisponde alla minima intensità e il segno posto dal paziente. Sulla base di diversi studi, sono stati suggeriti i seguenti valori cut-off: da 0 a 4 mm: "nessun dolore"; da 5 a 44 mm: "dolore lieve"; da 45 a 74 mm: "dolore moderato"; da 75 a 100 mm: "dolore severo".

Tornando allo studio della struttura del dataset, dopo aver elencato le diverse scale di rilevamento dell'intensità dolorifica, possiamo dire che partendo dal video effettuato per ogni singolo paziente, sono stati estratti dei frame, ed ogni frame è stato codificato con il sistema FACS e successivamente con l'intensità della scala PSPI (entrambi ampiamente discussi nel capitolo 3. Ogni azione facciale è descritta in termini di una delle 44 unità

di azione individuali (AU). Poiché esiste una notevole quantità in letteratura in cui la FACS è stata applicata all'espressione del dolore, solo le azioni che sono state implicate come possibilmente correlate al dolore sono state prese in considerazione, tra queste troviamo:

- abbassamento delle sopracciglia (AU4)
- sollevamento delle guance (AU6)
- sollevamento delle guance (AU6)
- serraggio delle palpebre (AU7)
- arricciatura del naso (AU9)
- sollevamento del labbro superiore (AU10)
- sollevamento del labbro obliquo (AU12)
- allungamento orizzontale delle labbra (AU20)
- apertura delle labbra (AU25)
- abbassamento della mascella (AU26)
- allungamento della bocca (AU27)
- chiusura degli occhi (AU43)

Ad eccezione di AU 43, ogni azione è stata inoltre codificata su una dimensione di intensità di 5 livelli (A-E) da uno dei tre codificatori certificati FACS. Le azioni sono state codificate frame by frame. Un esempio delle video sequenze su quattro pazienti del database è osservabile in figura 21.



Figura 21: Sequenze di pazienti affetti da dolore alla spalla presenti nel dataset di interesse

Nel database è inoltre presente una sezione dedicata allo sviluppo di un sistema basato su Active Appearance Model (AAM): un algoritmo di visione artificiale per abbinare un modello statistico di forma e aspetto di un oggetto a una nuova immagine. L'approccio AAM è stato utilizzato per tracciare il viso ed estrarre le caratteristiche facciali. Nella distribuzione dei dati vengono inclusi 66 punti di riferimento AAM per ogni immagine.

In figura 23 è disponibile una visione del funzionamento del sistema AAM.

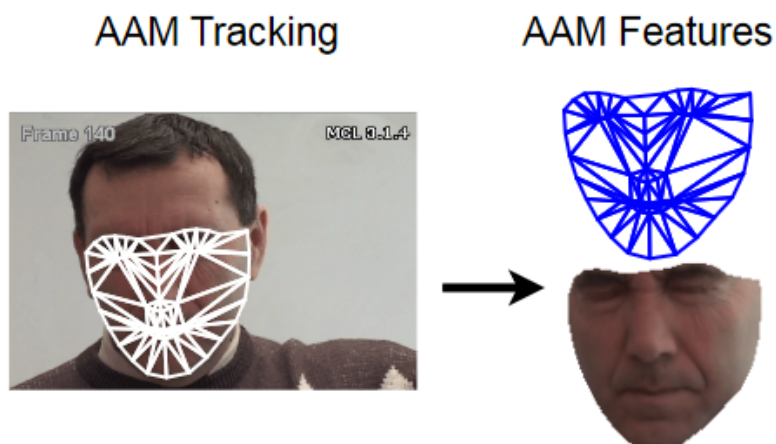


Figura 22: Esempio di tracciamento e feature extraction grazie al sistema AAM

Dall'intero UNBC-McMaster Pain Shoulder Archive disponibile, sono state preparate 200 sequenze da 25 soggetti diversi nella parte attiva del set di dati per la distribuzione alla comunità di ricerca. Da queste 200 sequenze c'è un totale di 48398 frame che sono stati codificati FACS e tracciati AAM. Tuttavia, il dataset risulta altamente sbilanciato, avendo dei casi di non dolore molto più elevati dei casi in cui il dolore veniva percepito anche in minima parte. Infatti, si può dire che l'83,6% dei frame aveva un punteggio PSPI di 0 e il 16,4% aveva frame in cui aveva un punteggio PSPI superiore ad 1. Questa conoscenza è stata fondamentale per studiare una direzione progettuale adeguata per un corretto svolgimento nel compito della pain detection, di cui discuteremo approfonditamente nel sotto - capitolo 4.2.

Per una chiarezza aggiuntiva, di seguito viene riportata la struttura del file system del database ottenuto, e le sue parti di interesse.








Nome	Ultima modifica	Tipo	Dimensione
 AAM_landmarks	17/06/2021 20:15	Cartella di file	
 Frame_Labels	17/06/2021 20:17	Cartella di file	
 Images	17/06/2021 20:11	Cartella di file	
 Sequence_Labels	21/09/2010 11:28	Cartella di file	
 Agreement_Form_2020.05.27	04/06/2020 15:13	Documento di Mic...	16 KB
 FG_PainArchive	10/10/2019 20:45	File PDF	3.619 KB
 README_File	10/10/2019 20:45	File PDF	113 KB

Figura 23: Struttura del database

Come si nota dall'immagine, all'interno della cartella sono contenuti:

- AAM\_landmarks: ovvero 66 punti di landmark utilizzati per estrarre features nel volto dei pazienti;
- Frame\_labels: una cartella contenente al suo interno altre due sotto-cartelle, una corrispondente alla codifica FACS frame by frame, l'altra alla codifica PSPI;
- Images: le immagini reali suddivise per pazienti e per sessioni, in formato "png";
- Sequence\_labels: una serie di scale di self-evaluation, discusse precedentemente;
- Agreement\_form: richiesta, in formato word, per l'ottenimento del database;
- FG\_PainArchive: articolo riportato anche in tabella 17 che spiega meglio la struttura del database, e l'approccio utilizzato dai creatori per compiere il task di pain detection;
- README\_file: un file pdf che spiega ogni singolo componente della presente cartella.

Nel dataset inoltre è presente una suddivisione aggiuntiva: ogni paziente è stato sottoposto a delle sessioni come detto precedentemente, dunque ogni cartella che rappresenta il paziente è composta da altre sotto - cartelle rappresentanti le rispettive sessioni. Nella figura 24 viene mostrata tale suddivisione.

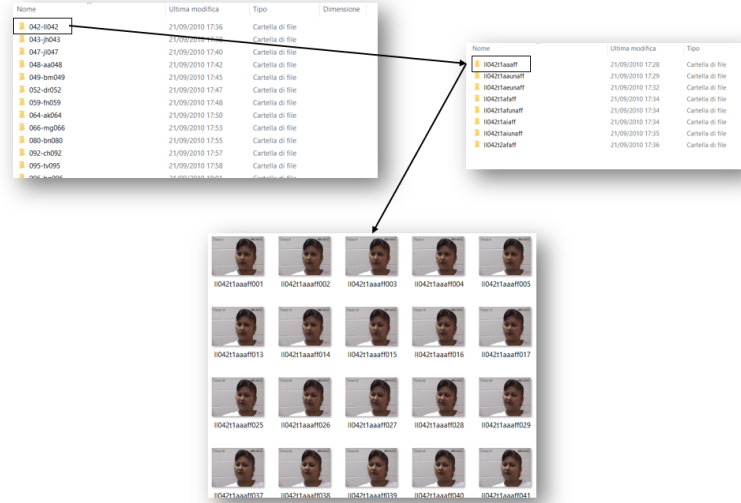


Figura 24: Struttura del database

Nella figura infatti, vengono mostrate in alto a sinistra le cartelle relative ad ogni singolo paziente (in totale 25), prendendo la prima si noti come questa si articola in altre sotto - cartelle relative alle sessioni ( in questo caso 8), per ogni sessione infine troviamo una serie di immagini corrispondenti ai frames.

Una volta discussa approfonditamente la struttura del set di dati si affronterà, nel capitolo successivo, la spiegazione dell'implementazione del presente lavoro.

## 4.2 Pre-processing delle immagini

In questa sezione mostreremo la pipeline di pre-processing delle immagini ottenute dal dataset precedentemente discusso. Tutta la procedura di implementazione è stata svolta in locale, usufruendo del linguaggio di programmazione Python ed utilizzando l'IDE Pycharm (Versione 2021.1.1).

In primo luogo è stato necessario comprendere la struttura del file system in cui era presente la cartella delle immagini da processare, ai fini di implementare un codice efficace.

Per una maggiore chiarezza inseriamo la suddetta pipeline in figura 25.

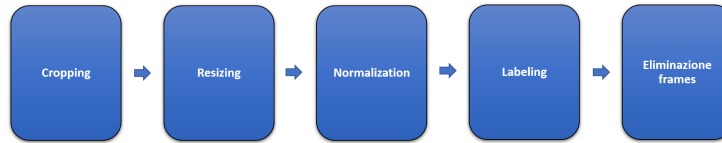


Figura 25: Pipeline di pre-processing analizzata in seguito

Come mostrato in figura, il primissimo step del pre-processing ha riguardato la procedura di taglio delle immagini: nel dataset i diversi frames presentavano molti elementi di rumore rispetto ai nostri scopi, per tale ragione è stato necessario individuare esclusivamente la parte di interesse, ovvero il volto, e ritagiarlo. Per compiere questo task si è deciso di utilizzare un algoritmo di rilevamento del volto, in particolare l'algoritmo di Viola - jones [14]. Nel codice 1 viene mostrata la parte di codice relativa alla fase di taglio delle immagini.

```

1 face_cascade = cv.CascadeClassifier('
    PATH_TO_CASCADE_CLASSIFIER_FOLDER')
2 f = 'PATH_TO_IMAGES'
3 sotto_car = os.listdir(f)
4 for patient in sotto_car:
5     p_path = f + "/" + patient
6     pat_path = os.listdir(p_path)
7     for session in pat_path:
8         s_path = p_path + "/" + session
9         ses_path = os.listdir(s_path)
10        for image in ses_path:
11            i_path = s_path + "/" + image
12            im=cv.imread(i_path)
13            gray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
14            faces = face_cascade.detectMultiScale(gray, 1.1, 4)
15            for (x, y, w, h) in faces:
16                cv.rectangle(im, (x, y), (x + w, y + h), (255,
255, 255), 2)
17                faces = im[y:y + h, x:x + w]
18                cv.imwrite(i_path, faces)

```

Listing 1: Fase di cropping delle immagini

Come si nota dal codice 1 è stato necessario seguire i seguenti step per ottenere un buon risultato:

- Caricamento del file contenente il classificatore pre-addestrato per il riconoscimento facciale. In questo step, avendo a disposizione molti file da poter scegliere, ci si è orientati verso il classificatore per facce poste frontalmente ovvero “l’haarcascade\_frontalface\_alt2.xml”;
- Caricamento delle immagini: in secondo luogo è stato necessario definire il percorso relativo per trovare le immagini e caricarle;
- Detection del volto e taglio: Successivamente è stata richiamata la funzione per rilevamento del volto, e disegnando un rettangolo sulla parte rilevata, è stata tagliata esclusivamente la parte di interesse.

Di seguito viene mostrato il risultato relativo ad un immagine dopo il primo step della pipeline, ovvero il taglio delle parti di interesse.



Figura 26: Esempio di applicazione dell’algoritmo di face-detection per il cropping dell’immagine

Come si vede in figura sono state tagliate tutte le parti che non erano rilevanti per i nostri scopi, lasciando esclusivamente il volto e le caratteristiche facciali.

Il secondo step ed il terzo step della pipeline sono stati: il ridimensionamento e la normalizzazione delle immagini(già tagliate). Il resizing è stato applicato ad ogni frame, portandolo alla dimensione fissa di : 224x224x3 pixel.

Tale dimensione è stata scelta in quanto, come si mostrerà nel sotto-capitolo inerente all’estrazione di features e classificazione (4.3), l’architettura



tura di rete pre-addestrata da noi selezionata, prevede un primo layer convoluzionale con input un'immagine RGB di 224x224 pixel. Per il ridimensionamento è stato utilizzato il modulo *Image* della libreria *pillow* di Python, mentre per lo step della normalizzazione è stata usata la libreria *OpenCv*.

La struttura di entrambi gli script non viene riportata poichè segue il procedimento del codice mostrato in figura 1.

Il penultimo step della suddetta pipeline, invece ha riguardato l'etichettatura da associare ad ogni frame. Come già spiegato nel sotto-capitolo riguardante la struttura del database 4.1, le immagini sono state tutte etichettate e l'associazione con la rispettiva etichetta è stata fatta in base al nome dell'immagine stessa. Dunque in questa fase è stato necessario far coincidere perfettamente i nomi dell'etichetta e dell'immagine, leggere il contenuto dell'etichetta e suddividere i frame in due categorie: no pain (nessun dolore) e pain (presenza di dolore).

Di seguito riportiamo lo script di interesse per il completamento di questo step.

```
1 dataim="PATH_TO_FRAMES"
2 dataala="PATH_TO_LABELS"
3 pain="PATH_TO_PAIN_FOLDER"
4 nopain="PATH_TO_NOPAIN_FOLDER"
5 empty="PATH_TO_EMPTY_FOLDER"
6
7 result=[]
8
9 for frames in os.listdir(dataim):
10     impath=dataim+"/"+frames
11     name1=os.path.basename(os.path.splitext(impath)[0])
12     for label in os.listdir(dataala):
13         txt_file = dataala + "/" + label
14         name2 = os.path.basename(os.path.splitext(txt_file)[0])
15         if name1 in name2:
16             print("yes")
17             print(name1 + " / & /" + name2)
18             f=open(txt_file,"r")
19             ele=f.read()
20             if os.stat(txt_file).st_size == 0:
21                 print("empty---file")
22                 sh.copy(impath, empty)
23             else:
24                 split=[ele[i:i + n] for i in range(0, len(ele),
25                     n)]
26                 print(split)
27                 for i in split:
28                     print(split[1])
29                     last=split[1]
```

```

29         if last == '0.0':
30             print("NO PAIN")
31             sh.copy(impath, nopain)
32         else:
33             print("PAIN")
34             sh.copy(impath, pain)
35         break

```

Listing 2: Fase di cropping delle immagini

Nel codice in figura 2 notiamo alcuni passi salienti:

- In primis vi è stata la procedura per l'estrazione del nome dell'immagine in questione senza estensione (quindi dividendo il nome dalla sua estensione) tramite il modulo *basename*;
- Dopo aver estratto il nome dell'immagine, all'interno dello stesso ciclo for si è applicata la stessa procedura anche per tutte le etichette (rappresentate da un file di testo), quindi è stato diviso il nome del file dalla sua estensione(.txt);
- Il terzo passo è stato controllare se il nome dell'immagine era presente nella cartella contenente tutti i nomi delle etichette precedentemente estratti;
- Successivamente a questo controllo, è stato necessario aprire il suddetto file di testo corrispondente al nome dell'immagine, e leggere il suo contenuto. Se il contenuto risultava con un indice PSPI = 0, allora l'immagine veniva posizionata nella cartella "no-pain", altrimenti nella cartella "pain".

Una volta completata la procedura del penultimo step di etichettatura, dunque siamo arrivati alla struttura da noi desiderata per svolgere una classificazione binaria. Il problema riportato anche precedentemente nel sotto-capitolo di descrizione del dataset (4.1) era il potenziale sbilanciamento di quest'ultimo: l'insieme di dati originali risulta suddiviso in maniera non equa, in quanto l'86.3% delle immagini presenta un PSPI=0, e solo il 16.4% un PSPI maggiore di 0, come riportato anche in [8]. Per tale ragione l'ultimo step di questa pipeline ha riguardato l'eliminazione di parte dei frame dalla cartella "no pain", per superare il suddetto sbilanciamento e procedere con una classificazione più realistica. L'eliminazione è avvenuta senza intaccare il numero di soggetti totali, ma prendendo solo un numero di frames da ognuno di essi. In totale il numero di frames ottenuti per cartella sono:

- PAIN: 8008 frames
- NO-PAIN: 8800 frames

Conclusa infine tutta la pipeline di pre-processing delle immagini, si è passati alla fase di estrazione di features facciali e classificazione della rete, che discuteremo nel capitolo successivo.

### 4.3 Estrazione delle features e classificazione

In questo capitolo andremo ad approfondire le tipologie di reti utilizzate per la fase di estrazione di features e di classificazione, descrivendo inoltre il tipo di validazione implementata in fase di addestramento.

L’approccio da noi proposto, come spiegato anche nel sotto-capitolo 3.2, si basa sulla tecnica del transfer learning + fine tuning, partendo da una rete pre-addestrata su un dataset di grandi dimensioni. In particolare le reti prese in considerazione in questo lavoro sono:

- VGG16: una rete neurale convoluzionale composta da 16 layer.
- ResNet50: una rete neurale convoluzionale composta da 50 layer.

**VGG16** VGG16 (figura 27) è un modello di rete convoluzionale neurale proposto per la prima volta nel 2014 da K. Simonyan e A. Zisserman dell’Università di Oxford all’interno dell’articolo “Very Deep Convolutional Networks for Large-Scale Image Recognition” [11]. L’architettura di questa CNN non è complessa, in quanto utilizza solamente una serie di strati convoluzionali 3x3 messi in sequenza, per un totale di 16 layers allenabili, intervallati da alcuni strati di max pooling, che gestiscono la riduzione dei volumi.

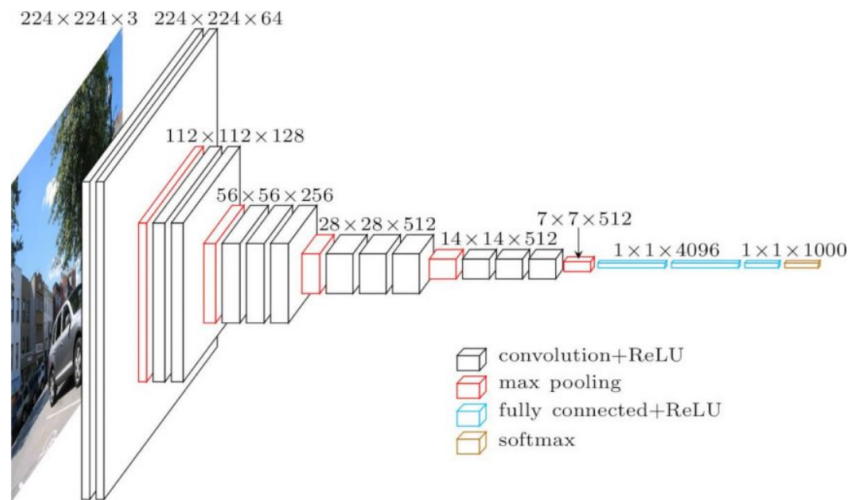


Figura 27: Architettura della rete VGG16

Nello specifico, la topologia di VGG-16 è composta dai seguenti strati:

- Strato convoluzionale (attivazione Relu) con 64 filtri (Conv1)
- Strato convoluzionale con 64 filtri + max pooling
- Strato convoluzionale con 128 filtri (Conv2)
- Strato convoluzionale con 128 filtri + max pooling
- Strato convoluzionale con 256 filtri
- Strato convoluzionale con 256 filtri (Conv3)
- Strato convoluzionale con 256 filtri + max pooling
- Strato convoluzionale con 512 filtri
- Strato convoluzionale con 512 filtri (Conv4)
- Strato convoluzionale con 512 filtri + max pooling
- Strato convoluzionale con 512 filtri
- Strato convoluzionale con 512 filtri (Conv5)
- Strato convoluzionale con 512 filtri + max pooling

- Strato fully-connected con 4096 nodi (fc6)
- Strato fully-connected con 4096 nodi (fc7)
- Strato di output con attivazione softmax con  $N$  nodi

Quindi, come evidenziato anche in figura 27, sono presenti 5 tipologie di layer:

- Layer convoluzionale: Questi layers rappresentano il fulcro delle ConvNets. Sono costituiti da blocchi tridimensionali di neuroni artificiali, connessi localmente ad una regione limitata dell'immagine in input e del filtro successivo. Ogni neurone calcola il prodotto scalare tra i propri pesi e la regione a cui è connesso utilizzando la formula 1 riportata in seguito:

$$Y = \sum (weight \times input) + bias \quad (1)$$

Il fatto che i neuroni della rete non siano totalmente interconnessi tra loro, come accade invece nelle reti neurali classiche, permette di gestire immagini molto grandi in input senza memorizzare un eccessivo numero di parametri.

- Layer di attivazione: Ogni layer di convoluzione è necessariamente seguito da uno strato di attivazione, tanto che molto spesso questi ultimi vengono considerati parte integrante degli strati convoluzionali. Lo scopo delle funzioni di attivazione è quello di fornire alla rete un comportamento non-lineare, in modo da permettere la risoluzione di problemi complicati, utilizzando un numero relativamente ridotto di nodi. Senza la presenza di una funzione di attivazione non lineare, una rete neurale si comporterebbe come una rete ad un solo strato, a prescindere dal numero effettivo di hidden-layers presenti.
- Layer di pooling: I layer di Pooling consentono di effettuare un'operazione di sotto-campionamento spaziale dei volumi in ingresso. Nel caso di un'immagine, il processo consiste nel sostituire i valori di una regione di pixel, di dimensione pari alla grandezza del filtro di pooling, con un unico pixel dal valore pari al massimo tra i pixel iniziali, nel caso di *max-pooling*, o alla loro media, nel caso di *average-pooling*. Nella rete VGG16, gli strati di max-pooling utilizzano filtri di dimensione 2x2.

- **Layer Fully-Connected:** Dopo la serie di strati di convoluzione e di max-pooling, la rete VGG-16 comprende due strati fully-connected (fc-layers) da 4096 nodi. Questi layer equivalgono a quelli delle classiche reti neurali artificiali, il loro comportamento è identico a quello degli strati di convoluzione, con l'unica differenza nel fatto che i nodi sono interamente connessi a tutti i neuroni degli strati precedenti e non solamente ad una piccola regione.
- **Layer di output:** Infine, l'output layer, anch'esso interamente connesso con il volume precedente, si occupa della classificazione vera e propria. Il numero di nodi presenti in questi layer dipende dal numero di classi di interesse del dataset corrispondente.

Nel lavoro qui presentato, come spiegheremo nel capitolo 4.3.1, si utilizzerà la rete VGG16 pre-allenata su due differenti dataset per avere una chiara valutazione comparativa dei risultati ottenuti.

**ResNet50** Il modello Resnet50 appartiene ad una famiglia di reti artificiali neurali conosciute come reti residuali (Residual Networks). Le CNNs profonde hanno dimostrato di poter risolvere molto accuratamente gran parte dei compiti di classificazione di immagini e riconoscimento visivo. Per questo motivo, negli ultimi anni, si è verificata una tendenza ad aumentare sempre di più la profondità delle reti. Tuttavia, più si va in profondità, più l'addestramento delle reti convoluzionali classiche diventa un compito lungo e complesso ed anche la precisione inizia a saturarsi o, peggio, a degradarsi. Le reti residuali si basano su un nuovo ed innovativo tipo di blocchi (detti residual blocks) e sul concetto di *residual learning* per cercare di risolvere entrambi i problemi.

I blocchi residuali (figura 28) sottopongono un input  $x$  ad una sequenza di operazioni di convoluzione ed attivazione, ottenendo un output  $F(x)$ , al quale verrà sommato lo stesso input ( $x$ ), detto residuo. Le Residual Networks hanno dimostrato di essere più semplici da allenare rispetto alle CNN classiche ed, inoltre, non vanno incontro ad un degrado delle performance all'aumentare della profondità della rete stessa.

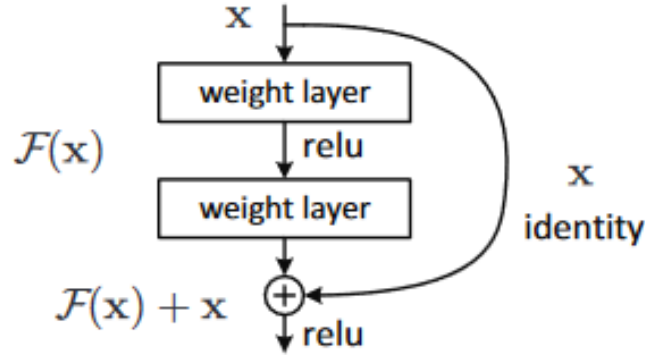


Figura 28: Esempio di residual block

Nello specifico la rete Resnet-50 utilizza due tipi di blocchi residuali (figura 29), il primo, detto blocco identità (a sinistra nell'immagine), si comporta come i blocchi residuali tradizionali descritti in precedenza, mentre il secondo, detto blocco di convoluzione, sottopone il tensore in input ad un diverso numero di strati convoluzionali in entrambi i rami, per poi sommarne i risultati.

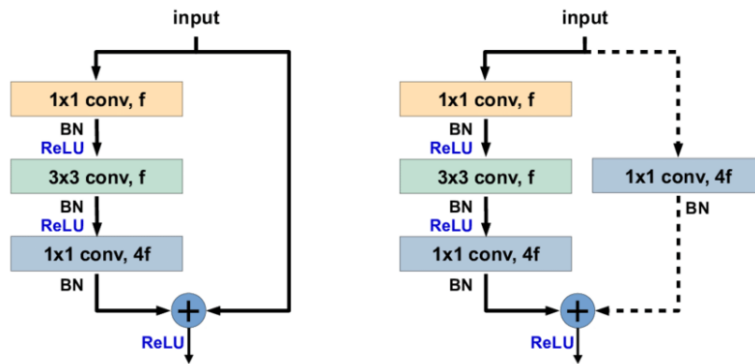


Figura 29: Blocco d'identità e blocco di convoluzione della ResNet-50

Nonostante l'architettura di Resnet-50 sia completamente differente rispetto a quella di VGG-16, il tipo di strati utilizzati è sostanzialmente analogo. L'unica differenza sta nel tipo di layer utilizzati per la regolarizzazione della rete: in questa architettura, infatti, gli strati di max-pooling sono stati

sostituiti da un nuovo tipo di layer, detto di *batch normalization*<sup>3</sup>. In figura 30 viene mostrata l'architettura della rete ResNet50.

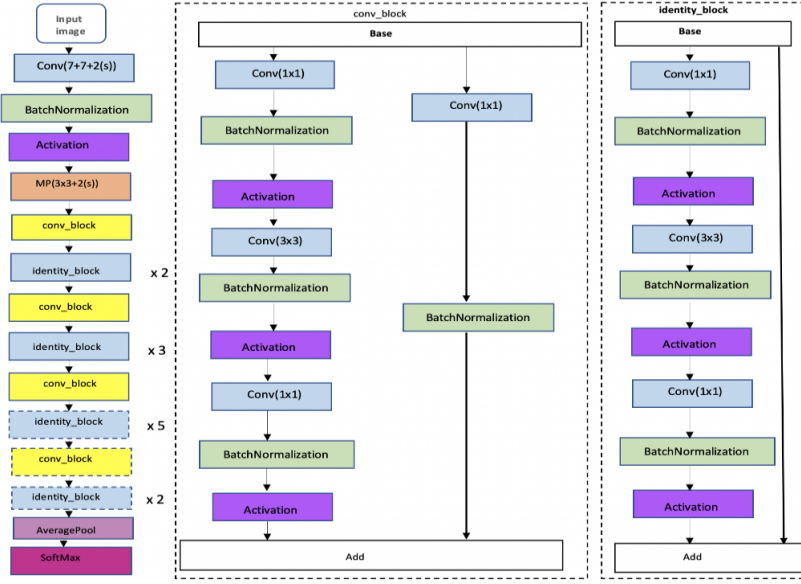


Figura 30: Struttura a blocchi della ResNet-50

Prima di parlare dell'implementazione delle reti utilizzate, spieghiamo nel dettaglio la tecnica di validazione utilizzata per testare le performance del modello: Per definire il grado di accuratezza o l'efficacia di un qualsiasi modello di machine learning è necessario eseguire una o più valutazioni sugli errori che si ottengono nelle previsioni.

In genere, dopo l'allenamento viene effettuata una stima dell'errore per il modello, meglio nota come valutazione dei residui. In questo processo, viene eseguita una stima numerica della differenza nelle risposte previste e originali, chiamata anche errore di addestramento (training error). Tuttavia, il problema con questa tecnica di valutazione è che non fornisce un'indicazione di come il modello di apprendimento generalizzerà su un set di dati.

Per ovviare a questo si può utilizzare la *cross validation*, o *convalida incrociata*, in particolare questa tecnica verrà utilizzata in tutti i modelli implementati.

<sup>3</sup> Questi layer si interpongono tra gli strati convoluzionali e quelli di attivazione e consentono la normalizzazione degli output ad ogni strato latente della rete



**Cross validation** La cross validation è un metodo statistico utilizzato per stimare l'abilità dei modelli di apprendimento automatico: in genere, dopo l'addestramento della rete viene effettuata una stima dell'errore per il modello stesso, meglio nota come valutazione dei residui. In questo processo viene eseguita una stima numerica della differenza tra le risposte previste e le risposte originali, chiamata anche errore di addestramento. Tuttavia, questo ci dà solo un'idea dell'efficacia del nostro modello sui dati utilizzati per addestrarlo. E' possibile però, che il modello stia sottoadattando o sovraadattando i dati. Quindi, il problema con questa tecnica di valutazione è che non dà un'indicazione di quanto bene il *learner* generalizzerà a un set di dati indipendente/invisibile. La tecnica di validazione incrociata, invece, aiuta a capire i problemi che si verificano in fase di addestramento e prevede di confrontare algoritmi di apprendimento dividendo i dati in due segmenti: uno utilizzato per apprendere o addestrare il modello e l'altro utilizzato per convalidarlo. Nella tipica convalida incrociata, i set di addestramento e validazione devono incrociarsi in cicli successivi in modo che ogni slot di dati abbia la possibilità di essere convalidato. La forma base della convalida incrociata è la convalida k-fold, che suddivide il dataset di training in k slot differenti da usare come set di convalida. Altri metodi di cross validation sono casi speciali di convalida incrociata k-fold.

In figura 31 vediamo un esempio di 5-fold cross validation.

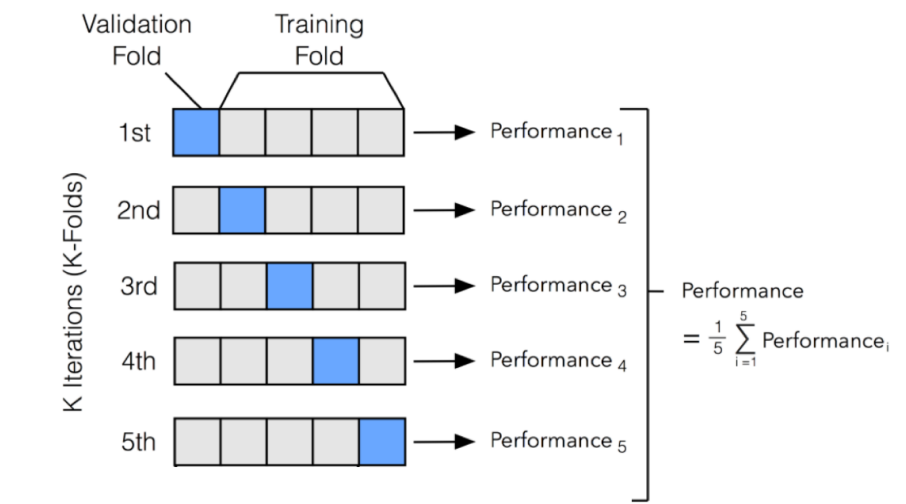


Figura 31: Esempio di una convalida incrociata con 5 slot di dati

Come notiamo dall'immagine, alla fine dell'addestramento le performance dei diversi slot vengono mediate per ottenere un valore più veritiero.

Esistono diverse tipologie di convalida incrociata. In questo articolo ne vediamo 5:

- Holdout: il tipo più semplice di convalida dei dati, con un'unica suddivisione.
- Leave One Out Cross Validation (LOOCV): in cui il numero delle suddivisioni è pari al numero delle osservazioni che abbiamo nel dataset.
- K-fold CV: la convalida più diffusa che permette di definire k suddivisioni del dataset.
- Stratified Cross Validation: in cui in ogni piega o suddivisione la distribuzione dei campioni tra le classi viene mantenuta costante.
- ShuffleSplit: un metodo ibrido tra il metodo holdout e la convalida k-fold.

nel nostro caso la metodologia usata è un misto tra *Stratified Cross Validation* e *ShuffleSplit*, per implementarla è stata utilizzata la libreria *Sklearn*<sup>4</sup> importando il modulo *StratifiedShuffleSplit* nel seguente modo:

```
1 from sklearn.model_selection import StratifiedShuffleSplit
```

Listing 3: Importare StratifiedShuffleSplit tramite la libreria Sklearn

La **Stratified cross validation** è una tecnica in cui riordiniamo i dati in modo tale che ogni suddivisione abbia una buona rappresentazione dell'intero set di dati. Obbliga ogni slice ad avere almeno  $m$  istanze di ogni classe. Questo approccio garantisce che una classe di dati non sia sovrappesata soprattutto quando la variabile target è sbilanciata. Inoltre, aiuta a ridurre sia il bias che la varianza. Mentre la tecnica **ShuffleSplit** è un ibrido tra la suddivisione tradizionale del set di training e il metodo di convalida incrociata k-fold. La differenza principale che la convalida ShuffleSplit ha con il k-Fold è che la prima può ottenere valori di test duplici. Il motivo di ciò è che la *ShuffleSplit* campionerà casualmente l'intero set di dati durante ogni iterazione per generare un set di allenamento e un set di test. Poiché

---

<sup>4</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.StratifiedShuffleSplit.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html)

si esegue il campionamento dell'intero set di dati durante ciascuna iterazione, i valori selezionati durante un'iterazione potrebbero essere nuovamente selezionati durante un'altra iterazione.

Vengono create divisioni casuali dei dati nel set di test e di allenamento e quindi viene ripetuto il processo di suddivisione e valutazione dell'algoritmo più volte, proprio come nel metodo di convalida incrociata.

Tornando all'implementazione delle reti utilizzate, come accennato precedentemente, nel presente lavoro abbiamo preso in considerazione tre tipi di modelli pre-addestrati, comparando i diversi risultati per valutare le rispettive performance, in particolare faremo riferimento a:

- rete VGG-16 pre-addestrata su *ImageNet*[4]: Questo particolare modello utilizza la rete precedentemente discussa (VGG-16) allenata su un'ampia base di dati contenente 14 milioni di immagini che sono state annotate manualmente con l'indicazione degli oggetti in esse rappresentati. ImageNet è stato realizzato per l'utilizzo, in ambito di visione artificiale, nel campo del riconoscimento di oggetti. Di questo modello si testeranno due versioni successivamente discusse, a cui faremo riferimento tramite i seguenti nomi:
  - `Pain_detection_model1_version1` (capitolo 4.3.1.1);
  - `Pain_detection_model1_version2` (capitolo 4.3.1.2).
- rete VGG-16 pre-addestrata su *VGGface*[10]: un modello che utilizza VGG-16 allenata su un ampio dataset di riconoscimento facciale creato dallo stesso gruppo di ricerca per classificare i volti appartenenti a 2622 individui. Il presente DB è stato scelto con lo scopo di misurare le performance della rete VGG16 in presenza di un pre-addestramento su un set di dati contenenti *esclusivamente* volti, rispetto ad ImageNet. VGGFace creato ed implementato dalla Visual Geometry Group dell'università di Oxford, contiene circa 2.26 milioni di facce prese dal web. Anche in questo caso, le versioni testate sono due:
  - `Pain_detection_model2_version1` (capitolo 4.3.2.1);
  - `Pain_detection_model2_version2` (capitolo 4.3.2.2).
- rete ResNet-50 pre-addestrata su *VGGface2* [1]: rete convoluzionale discussa in precedenza, allenata su una versione più recente del dataset VGGFace. Tale versione contiene un totale di 3,31 milioni di facce prese dal web. Di questo modello abbiamo testato solo una versione, spiegando poi la motivazione di tale scelta:

– Pain\_detection\_model3\_version1 (capitolo 4.3.3.1).

Nel prossimo capitolo entreremo nello specifico di ogni rete appena discussa, evidenziando la struttura ed i parametri utilizzati per la progettazione.

#### 4.3.1 Primo modello: rete VGG-16 pre-addestrata su dataset ImageNet

Il primo modello che abbiamo testato, come riportato precedentemente, è la rete VGG-16 pre-addestrata su ImageNet, di questa prima prova verranno implementati due versioni, trattate successivamente.

Per la parte di estrazione di features e classificazione (capitolo class) è stato utilizzato il tool Google Colab<sup>5</sup> usufruendo del linguaggio di programmazione Python.

In primo luogo per testare il primo modello è stato necessario trasformare le immagini precedentemente pre-processate in *Numpy arrays* per ottimizzare la gestione dei dati stessi. In secondo luogo abbiamo deciso di compiere un altro tipo di etichettatura: essendo la classificazione di interesse una classificazione binaria, abbiamo etichettato le classi “pain” e “no pain” con le rispettive etichette “1” e “0”. Il codice riportato di seguito in figura 4 rappresenta questi due task.

```
1 from PIL import Image
2 from numpy import asarray
3 import numpy as np
4 import os
5
6 path_dol = "Path_to_first_files"
7 path_out="Path_to_second_files"
8
9 for im in os.listdir(path_dol):
10     open_im=Image.open(os.path.join(path_dol,im))
11     im_array=asarray(open_im)
12     file_split=os.path.splitext(im)
13     filename=file_split[0]
14     data_out=path_out + filename + ".npy"
```

Listing 4: Codice per convertire le immagini in numpy arrays

Come si nota dal codice, è stato necessario ripetere la procedura per entrambe le classi, mantenendo tale suddivisione invariata.

---

<sup>5</sup>[https://colab.research.google.com/notebooks/intro.ipynb?utm\\_source=scs-index#recent=true](https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index#recent=true)

```

1 from PIL import Image
2 from numpy import asarray
3 import numpy as np
4 import os
5
6 path = "path_to_numpy_img"
7
8 X = []
9 Y = []
10
11 dir = path
12 classes = ["DOLORE", "NON DOLORE"]
13 classes_list = os.listdir(dir)
14 classes_list.sort()
15
16 full_path_DOL = os.path.join(dir, classes_list[1])
17 samples_DOL = os.listdir(full_path_DOL)
18 samples_DOL.sort()
19 for a in samples_DOL:
20     b = os.path.join(full_path_DOL, a)
21     data = np.load(b)
22     X.append(data)
23     y = [0]*len(classes_list)
24     y[1] = 1
25     Y.append(y)

```

Listing 5: Etichettatura immagini con labels "0" ed "1"

Anche nel codice in figura 4 è stato necessario implementare codici distinti per entrambe le classi, i passi affrontati sono stati i seguenti:

- Importare le librerie di interesse: Pillow <sup>6</sup>, numpy <sup>7</sup> per la gestione degli array, ed os<sup>8</sup> per navigare nel file system;
- Definire di due liste vuote: per ospitare i numpy array di interesse;
- Definire un ciclo *for*: per caricare i numpy array e generare le etichette per le classi binarie.

Una volta affrontati i precedenti step, vi è stata l'effettiva realizzazione delle reti da testare, anche qui dapprima abbiamo importato tutte le librerie di interesse con il seguente codice:

<sup>6</sup> <https://pillow.readthedocs.io/en/stable/>

<sup>7</sup> <https://numpy.org/>

<sup>8</sup> <https://docs.python.org/3/library/os.html>

```

1 from keras.models import Sequential
2 from keras import layers
3 import tensorflow as tf
4 from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
5   , Dense, Activation, GlobalMaxPooling2D, LSTM
6 from keras import applications
7 from keras.preprocessing.image import ImageDataGenerator
8 from keras import optimizers
9 from tensorflow.keras.applications.vgg16 import VGG16
10 from keras.models import Model

```

Listing 6: Importazione delle librerie necessarie per implementare le reti

Come si nota, abbiamo utilizzato la libreria di Python *Keras* per la definizione delle reti e dei corrispondenti layers, ed inoltre abbiamo importato dal modulo *applications* la rete testata inizialmente, e discussa in questo capitolo: la VGG16. In secondo luogo abbiamo definito i parametri della rete, utilizzando il seguente codice.

```

1 image_size = 224
2 input_shape = (image_size, image_size, 3)
3
4 pre_trained_model = VGG16(input_shape=input_shape, include_top=
   False, weights="imagenet")

```

Listing 7: Definizione dei parametri della VGG16

Analizzando nel dettaglio il codice in figura 7, abbiamo:

- Dichiarato la variabile *image\_size* istanziandola a 224, ovvero il valore della dimensione delle immagini precedentemente pre-processate
- Definito *input\_shape*, ovvero l'input alla nostra rete
- Istanziato la VGG16, passando come parametri:
  - *input\_shape*;
  - *Include\_top* impostato su *False* poichè, essendo la nostra implementazione finalizzata alla tecnica del fine tuning (discussa nel capitolo 3.2) gli strati di rete finali non dovevano essere inclusi;
  - *weights* impostato su *imagenet*, ovvero i pesi da caricare come pre-addestramento, dal dataset di interesse.

Per dare un'idea ancora più chiara della rete testata mediante tecnica del fine tuning, in figura 32 è mostrata l'architettura della VGG-16 escludendo il “*top*” della rete stessa, come i parametri da noi impostati.

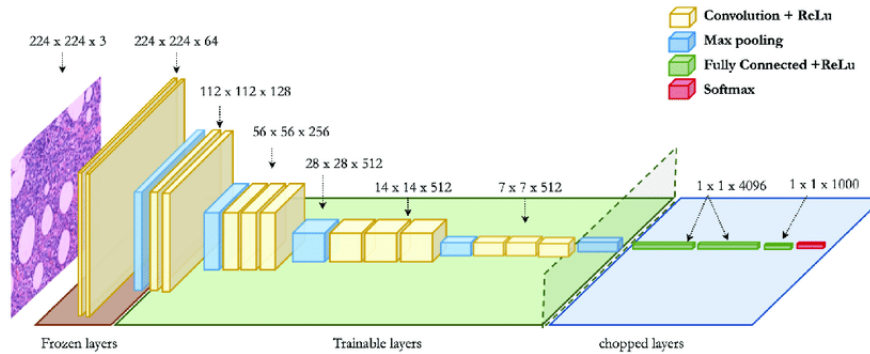


Figura 32: Fine tuning sulla VGG con *include\_top* impostato su *False*

dall'immagine si nota come, che si “tagliano” gli strati *Fully connected* finali, lasciando esclusivamente tutti gli strati fino all'ultimo layer di *Max pooling*.

#### 4.3.1.1 Pain\_detection\_model1\_v1

In questo primo sotto-capitolo, spiegheremo in dettaglio la prima versione del primo modello implementato. Il primo step in questa versione è stato rendere tutti gli strati presenti nella VGG-16 (mostrati in figura 32) come layer *non* allenabili (in gergo “congelati”), quindi immagazzinare i pesi della rete pre-addestrata senza necessità di includerli nel training dell'architettura. Per compiere questo task è stato sviluppato il seguente codice:

```

1 for layer in pre_trained_model.layers:
2     layer.trainable = False
3
4 for layer in pre_trained_model.layers:
5     print(layer, layer.trainable)
6
7 pre_trained_model.summary()
```

Listing 8: Freezing degli strati dedicati all'estrazione delle features della VGG-16

In questa prima porzione di codice, vediamo che il parametro *layer.trainable* è stato settato a *False*, come precedentemente spiegato. Inoltre per avere una verifica degli strati addestrabili, ed un riassunto dei layer totali della presente rete, abbiamo stampato la struttura finale della VGG-16, tramite il comando di *printing*.

L'output stampato dal *print*, risulta il seguente:

```
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0x7f4e03a81b10> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4e02e308d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4e025e5310> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f4e025ba550> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03bd850> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03d1310> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f4df03d3710> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03d5fd0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03dbd90> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03d3350> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f4df0368050> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df036b8d0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df0368390> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df03bdb90> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f4e02e75150> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df036f510> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4e025e4290> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f4df0368b10> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f4df03dfe50> False
Model: "vgg16"
```

Figura 33: Risultato del comando di *print* per layer addestrabili



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 224, 224, 3]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
Total params: 14,714,688		
Trainable params: 0		
Non-trainable params: 14,714,688		

Figura 34: Risultato del comando di *print* del riassunto dell'architettura

Come si nota in figura 33, tutti i layer precedentemente descritti della rete sono stati impostati con *non* addestrabili. Inoltre in figura 34 troviamo i nomi di tutti gli strati della VGG-16 seguiti da:

- Output shape: il tensore in uscita dal layer corrispondente che rappresenta un array a quattro argomenti: il primo è il *batch size*, un parametro che verrà impostato successivamente alla costruzione della rete, di cui parleremo di seguito; gli altri argomenti rappresentano la dimensione dell'immagine. Nel nostro caso la dimensione è 224x224x3 inizial-

mente, poi scendendo in profondità l'immagine subirà delle variazioni di dimensioni.

- Param #: mostra il numero di parametri addestrati per ogni livello.

Notiamo infatti che il totale di parametri che *non* sono addestrati è proprio il totale di parametri presenti nella VGG-16.

Definita l'architettura generale della rete vi è stata la parte di costruzione della rete specifica da testare, ed avendo utilizzato una tecnica di *Stratified-ShuffleSplit*, come metodo di *cross validation*, abbiamo validato per ogni split i nostri risultati includendo il tutto in un unico ciclo *for*. In particolare abbiamo riservato l'80% del dataset totale come dati di training, ed il 20% come dati di test. Di tutti i dati di training circa il 20% è stato utilizzato per la validazione (quindi 16% del dataset totale). In questa prima versione abbiamo aggiunto alla VGG-16 due layer *Fully connected* in cascata con funzione di attivazione *Relu* e *Softmax* rispettivamente, intervallati da uno strato di *Dropout* con rate al 50%;, come segue:

```
1
2 for train, test in cv.split(X, Y):
3
4     last_layer = pre_trained_model.get_layer('block5_pool')
5     last_output = last_layer.output
6     x = tf.keras.layers.GlobalMaxPooling2D()(last_output)
7     x = tf.keras.layers.Dense(512, activation='relu')(x)
8     x = tf.keras.layers.Dropout(0.5)(x)
9     x = tf.keras.layers.Dense(2, activation='softmax')(x)
10
11     model = tf.keras.Model(pre_trained_model.input, x)
12     model.compile(loss='categorical_crossentropy',
13                   optimizer=optimizers.SGD(learning_rate=0.001,
14                                             momentum=0.9),
15                   metrics=['accuracy'])
16
17     earlystop=tf.keras.callbacks.EarlyStopping(patience=5)
18     callbacks=[earlystop]
19     history=model.fit(x=X[train], y=Y[train], validation_split
20                      =0.2, epochs=50, batch_size=128, shuffle=True, callbacks=
21                      callbacks)
```

Listing 9: costruzione della rete ed aggiunta di due layer densi

Spighiamo a questo punto le parti più salienti del codice appena illustrato:

- *tf.keras.layers*: tramite questo comando siamo stati in grado di aggiungere strati alla VGG-16, in particolare gli strati aggiunti alla testa dell'architettura sono:
  - *GlobalMaxPooling* descritto nel capitolo 4.3 che prende in ingresso l'output dell'ultimo layer della VGG-16.
  - *Dense* , ossia uno strato fully connected composto da 512 nodi e funzione di attivazione *relu*.
  - *Dropout*: il layer di dropout è un layer che serve per la regolarizzazione, questa tecnica prevede di applicare il solito procedimento di training, togliendo prima una certa percentuale di neuroni in ogni hidden layer. Per ogni epoca di allenamento si sceglie (casualmente) quali neuroni tenere e quali scartare e si allena la rete così ottenuta. Si ripete quindi il procedimento, tenendo e scartando neuroni diversi ad ogni epoca: una volta che si ritiene che la rete sia pronta, si prende la rete originale e si aggiustano i pesi uscenti dai neuroni nascosti. In questa versione il rate (frequenza) di dropout è stata impostata a 0.5.
  - *Dense*: aggiunta di un altro strato fully connected corrispondente al numero di classi di interesse , con funzione di attivazione *softmax*.
- *tf.keras.Model*: con questo comando stiamo costruendo il modello, passando come parametro di input della rete, la VGG-16 e collegando i layer appena descritti in testa alla rete stessa.
- *model.compile*: nel comando di model compile passiamo diversi parametri di training della rete tra cui:
  - *loss*: Le funzioni di perdita vengono utilizzate per determinare l'errore (noto anche come “la perdita”) tra l'output dei nostri algoritmi e il valore target specificato. In parole povere, la funzione di perdita esprime quanto lontano dalla previsione è il nostro output calcolato. In questo caso è stata scelta una funzione specifica definita “categorical cross entropy” (chiamata anche perdita logaritmica), ovvero ogni probabilità di classe prevista viene confrontata con l'output 0 o 1 desiderato della classe effettiva e viene calcolato un punteggio / perdita che penalizza la probabilità in base alla distanza dal valore atteso effettivo. La penalità è di natura logaritmica e produce un punteggio elevato per grandi

differenze vicine a 1 ed un punteggio basso per piccole differenze tendenti a 0.

- *optimizer*: sono classi o metodi utilizzati per modificare gli attributi del modello di machine/deep learning come pesi e *learning rate* al fine di ridurre le perdite. Gli ottimizzatori aiutano a ottenere risultati più velocemente. Durante l'addestramento del modello, ottimizziamo i parametri e i pesi della rete stessa per ridurre al minimo la perdita e cerchiamo di rendere la nostra accuratezza di previsione il più corretta possibile. Ora per modificare questi parametri entra in gioco il ruolo dell'ottimizzatore, che lega i parametri del modello con la *loss function* aggiornando il modello in risposta all'output di quest'ultima. Nel nostro caso specifico, come ottimizzatore è stato scelto *SGD* (Stochastic gradient descent) esso esegue un aggiornamento dei parametri per ciascun esempio di addestramento ed esegue calcoli ridondanti per set di dati più grandi, poiché ricalcola i gradienti per lo stesso esempio prima di ogni aggiornamento dei parametri. Esegue aggiornamenti frequenti con un'elevata varianza che fa fluttuare pesantemente la funzione obiettivo. Fra gli argomenti del nostro ottimizzatore sono stati impostati:
  - \* il *learning rate*: è un parametro di ottimizzazione dell'algoritmo, che determina la dimensione del passo ad ogni iterazione durante lo spostamento verso il minimo della funzione di perdita. In questo caso è stato impostato a 0.001;
  - \* il *momentum*: è un parametro che esprime la media mobile dei gradienti. Lo usiamo quindi per aggiornare il peso della rete; esso aiuta ad accelerare i gradienti nella giusta direzione.
- *metrics*: è una funzione che viene utilizzata per giudicare le prestazioni del modello, nel nostro caso è stata scelta la metrica dell'accuratezza (spiegata in dettaglio nel capitolo 5).
- *earlystop*: un parametro utilizzato per stoppare il training della rete in anticipo: interrompe l'allenamento quando una metrica monitorata ha smesso di migliorare. Esso è un parametro che serve per evitare il problema dell'*overfitting* dell'architettura di rete complessiva. L'argomento *patience* passato serve per definire il numero di epoche senza miglioramento dopo le quali l'allenamento verrà interrotto.
- *model.fit*: il comando `model.fit` è stato utilizzato per inizializzare il training vero e proprio della rete, fra i parametri settati troviamo

- $x=X$ : dati di input di training
- $y=Y$ : dati target di training
- *validation\_split*: parametro per definire la percentuale di divisione del dataset utile per la validazione. Nel nostro caso 20% del set di train (ossia circa 16% dell'intero dataset)
- *epochs*: Numero di epoche per ogni split
- *Shuffle*: Un booleano utilizzato per mescolare i dati di addestramento prima di ogni epoca
- *batch\_size*: Numero di campioni per l'aggiornamento gradiente
- *callbacks*: In questo caso è stata richiamata la funzione di *early-stop* precedentemente discussa

Come accennato prima per ogni split sono state stampate le rispettive metriche e la rispettiva matrice di confusione (capitolo 5) ed alla fine di ogni allenamento sono state stampate le metriche medie raggiunte dal modello testato.

#### 4.3.1.2 Pain\_detection\_model1\_v2

Nella presente versione invece è stato deciso di aggiungere maggiore complessità al modello appena descritto (4.3.1.1). Ciò è derivato dal fatto che la rete precedente presentava un leggero *underfitting* (problema che si verifica quando un modello non è in grado di acquisire il pattern sottostante dei dati). Per tale motivazione l'architettura di base è rimasta sempre la stessa (ovvero la rete mostrata in figura 32), rendendo nuovamente tutti i layer della rete *non* allenabili (figura 33) aggiungendo in testa all'architettura due strati convoluzionali (oltre ai due strati *Dense*), con lo scopo di aumentare la complessità della rete e rendere il modello un pò meno semplice (diminuendo conseguentemente l'underfitting complessivo).

Il seguente script mostra il codice della presente versione:

```

1 last_layer = pre_trained_model.get_layer('block5_pool')
2 last_output = last_layer.output
3 x = tf.keras.layers.Conv2D(512,2)(last_output)
4 x = tf.keras.layers.Conv2D(512,2)(x)
5 x = tf.keras.layers.GlobalMaxPooling2D()(x)
6 x = tf.keras.layers.Dense(512, activation='relu')(x)
7 x = tf.keras.layers.Dropout(0.5)(x)
8 x = tf.keras.layers.Dense(2, activation='softmax')(x)
9
10 model = tf.keras.Model(pre_trained_model.input, x)

```

```

11 model.compile(loss='categorical_crossentropy',
12               optimizer=optimizers.SGD(learning_rate=0.001,
13                                         momentum=0.9),
14               metrics=['accuracy'])
15
16 earlystop=tf.keras.callbacks.EarlyStopping(patience=5)
17 callbacks=[earlystop]
18 history=model.fit(x=X[train], y=Y[train], validation_split
19                  =0.2, epochs=50, batch_size=128, shuffle=True, callbacks=
20                  callbacks)

```

Listing 10: Costruzione delle rete con aggiunta di layer convoluzionali per aumentare la complessità

Come si nota nel codice appena proposto, l'unica differenza con l'architettura della precedente versione (versione 1), è l'aggiunta di due strati convoluzionali prima del layer di *pooling*:

- *tf.keras.layers.Conv2D*: il layer convoluzionale è uno strato particolare che serve per processare le immagini in input della rete. Esso è composto da una matrice chiamata kernel che rappresenta un filtro. Tale matrice è in generale di dimensioni minori rispetto all'immagine e viene applicata traslando il kernel su di essa, come indicato nella figura 35.

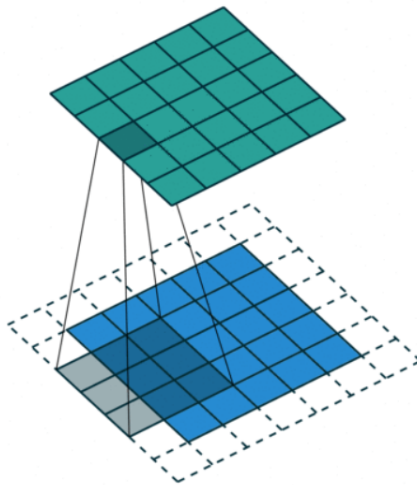


Figura 35: Esempio di operazione di convoluzione con kernel di dimensione 3x3

Il kernel in questo caso è la matrice 3 x 3 grigia che si sposta sulla immagine 5 x 5 blu. Il risultato è la matrice 5 x 5 verde, detta *feature map*. Il bordo bianco attorno all'immagine (chiamato *padding*), solitamente si utilizza per evitare che i bordi della immagine tendano a “sparire” nei layer successivi, ma si tratta di un parametro personalizzabile e quindi adattabile ad ogni situazione. Il risultato del filtro convoluzionale dipende ovviamente dal contenuto numerico della matrice del kernel, in pratica attraverso i kernel convoluzionali possiamo ottenere una infinità di filtri i quali possono evidenziare alcune caratteristiche dell'immagine e nascondere altre (ad esempio un kernel può evidenziare i contorni orizzontali, un altro quelli verticali, un terzo evidenziare solo le parti circolari, e così via). Nel nostro lavoro abbiamo aggiunto il suddetto layer passando come argomenti:

- *filters*: la dimensionalità dello spazio di output (cioè il numero di kernel da applicare all'immagine) in questo caso 512;
- *kernel\_size*: specifica l'altezza e la larghezza della finestra di convoluzione 2D.

Grazie all'aggiunta di due layer *Conv2D*, il numero di parametri è aumentato, osservando una marcata diminuzione dell'underfitting come spiegheremo nel dettaglio nei capitoli 5 e 6.

### 4.3.2 Secondo modello: rete VGG-16 pre-addestrata su dataset di volti

La seconda architettura testata è stata implementata utilizzando un altro dataset di pre-addestramento. L'*ImageNet*, discusso precedentemente, infatti, risulta essere un insieme di dati contenenti immagini di oggetti disparati e non solo volti. Per tale motivazione abbiamo pensato di allenare la stessa architettura di rete (ovvero la VGG-16) partendo però da un database che contenesse *esclusivamente* volti. Dopo un attenta analisi dei dataset disponibili secondo questo criterio, è stato scelto il database costruito dalla Vision Group, come spiegato anche nel sotto-capitolo 4.3, le versioni del dataset sono principalmente due:

- VGGface: prima versione del dataset contenente 2.26 milioni di immagini di volti, prese dal web. Insieme al DB i creatori hanno messo a disposizione anche un rete pre-addestrata sullo steso (la VGG-16).

- VGGface 2: seconda versione più recente del DB, contenente 3.31 milioni di immagini di volti. Anche qui i creatori hanno messo a disposizione due tipologie di rete pre-addestrate: La ResNet-50 e la SEnet (Squeeze-and-Excitation Network).

Dunque in questo secondo modello prenderemo in considerazione la VGG-16 pre-allenata sul dataset VGGface (prima versione). La metodologia iniziale segue quella descritta nel precedente modello, ovvero:

- Conversione delle immagini in array *Numpy*, come mostrato nel codice 4
- Etichettatura delle immagini con labels “0” ed “1” come mostrato nel codice 5

Dopo questi primi step è stato necessario importare le librerie, in questo caso però, è stata importata una libreria in più rispetto al modello precedente: la libreria Vggface, necessaria per caricare i pesi della VGG-16 pre-allenata. Quindi il codice di importazione delle librerie è il seguente:

```

1 from keras.models import Sequential
2 from keras import layers
3 import tensorflow as tf
4 from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
  , Dense, Activation, GlobalMaxPooling2D, LSTM
5 from keras import applications
6 from keras.preprocessing.image import ImageDataGenerator
7 from keras import optimizers
8 from tensorflow.keras.applications.vgg16 import VGG16
9 from keras.models import Model
10 from keras_vggface.vggface import VGGFace

```

Listing 11: Importazione delle librerie comprensiva di VGGface

Di cui vediamo che la differenza principale con il codice del primo modello è l'ultima riga, in cui abbiamo importato la VGGface richiamando il modulo *keras\_vggface.vggface*.

Una volta definite le librerie è stata caricata la rete con il seguente comando:

```

1 image_size = 224
2 input_shape = (image_size, image_size, 3)
3
4 pre_trained_model= VGGFace(input_shape=input_shape, include_top
  =False)

```

Listing 12: Caricamento della rete VGG-16 pre-addestrata su VGGface



Anche nel codice 12 abbiamo settato i parametri di input dell'architettura, definendo in fine la tipologia di rete di interesse: richiamando il comando *VGGFace*<sup>9</sup> abbiamo potuto caricare i pesi, definendo l'*input\_shape* ed escludendo la testa della rete (*include\_top= False*).

Anche per questo secondo modello, abbiamo previsto due tipi di versioni da testare:

- *Pain\_detection\_model2\_v1*: prima versione senza aggiunta di complessità;
- *Pain\_detection\_model2\_v2*: seconda versione con aggiunta di parametri per aumentare la complessità del modello.

#### 4.3.2.1 *Pain\_detection\_model2\_v1*

Come accennato precedentemente, questa prima versione è stata sviluppata senza aggiunta di complessità nell'architettura generale. In primo luogo è stato necessario mantenere i pesi della rete pre-addestrata intatti; per far questo si è deciso di “congelare” tutti gli strati della VGG16 deputati all'estrazione delle features, impostandoli come “non allenabili”, come mostrato nel codice che segue.

```
1 for layer in pre_trained_model.layers:
2     layer.trainable = False
3
4 for layer in pre_trained_model.layers:
5     print(layer, layer.trainable)
6
7 pre_trained_model.summary()
```

Listing 13: Freezing degli strati della VGG-16 pre-addestrata su VGGface

Il risultato dell'operazione di *printing* è mostrato nelle figure 36 e 37

---

<sup>9</sup> N.B: richiamando esclusivamente il comando “*VGGFace*” la rete che viene presa in considerazione è la VGG-16, in quanto è l'unica rete pre-addestrata sul presente dataset

```
<keras.engine.input_layer.InputLayer object at 0x7f78204a99d0> False
<keras.layers.convolutional.Conv2D object at 0x7f78204b4f10> False
<keras.layers.convolutional.Conv2D object at 0x7f7820b611d0> False
<keras.layers.pooling.MaxPooling2D object at 0x7f78102974d0> False
<keras.layers.convolutional.Conv2D object at 0x7f781029ca10> False
<keras.layers.convolutional.Conv2D object at 0x7f78102a4610> False
<keras.layers.pooling.MaxPooling2D object at 0x7f78102aeb90> False
<keras.layers.convolutional.Conv2D object at 0x7f78102a4390> False
<keras.layers.convolutional.Conv2D object at 0x7f7810236250> False
<keras.layers.convolutional.Conv2D object at 0x7f781023f750> False
<keras.layers.pooling.MaxPooling2D object at 0x7f7810242350> False
<keras.layers.convolutional.Conv2D object at 0x7f781023ab50> False
<keras.layers.convolutional.Conv2D object at 0x7f7810246f50> False
<keras.layers.convolutional.Conv2D object at 0x7f7810253350> False
<keras.layers.pooling.MaxPooling2D object at 0x7f781024b390> False
<keras.layers.convolutional.Conv2D object at 0x7f781025c6d0> False
<keras.layers.convolutional.Conv2D object at 0x7f781025fc10> False
<keras.layers.convolutional.Conv2D object at 0x7f7810269fd0> False
<keras.layers.pooling.MaxPooling2D object at 0x7f781025c410> False
Model: "vggface_vgg16"
```

Figura 36: Riassunto di strati addestrabili e non addestrabili della VGG-16 pre-allenata su VGGFace

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[ (None, 224, 224, 3) ]	0
conv1_1 (Conv2D)	(None, 224, 224, 64)	1792
conv1_2 (Conv2D)	(None, 224, 224, 64)	36928
pool1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2_1 (Conv2D)	(None, 112, 112, 128)	73856
conv2_2 (Conv2D)	(None, 112, 112, 128)	147584
pool2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv3_1 (Conv2D)	(None, 56, 56, 256)	295168
conv3_2 (Conv2D)	(None, 56, 56, 256)	590080
conv3_3 (Conv2D)	(None, 56, 56, 256)	590080
pool3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv4_1 (Conv2D)	(None, 28, 28, 512)	1180160
conv4_2 (Conv2D)	(None, 28, 28, 512)	2359808
conv4_3 (Conv2D)	(None, 28, 28, 512)	2359808
pool4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv5_1 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_2 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_3 (Conv2D)	(None, 14, 14, 512)	2359808
pool5 (MaxPooling2D)	(None, 7, 7, 512)	0
Total params: 14,714,688		
Trainable params: 0		
Non-trainable params: 14,714,688		

Figura 37: Riassunto di tutti gli strati presenti nella VGG-16 pre-addestrata su VGGFace

Una volta definita la struttura della rete pre-addestrata, esattamente come la versione del modello 1, sono stati aggiunti in coda due strati *Dense* già discussi in precedenza intervallati da un layer di Dropout.

```
1 last_layer = pre_trained_model.get_layer('pool5') # cambia solo
   il nome dell'ultimo layer della VGGFace
2 last_output = last_layer.output
3 x = GlobalMaxPooling2D()(last_output)
4 x = Dense(512, activation='relu')(x)
5 x = Dropout(0.5)(x)
6 x = Dense(2, activation='softmax')(x)
7
8 model = Model(pre_trained_model.input, x)
9 model.compile(loss='categorical_crossentropy',
10               optimizer=optimizers.SGD(learning_rate=0.001,
11                                         momentum=0.9),
12               metrics=['accuracy'])
13
14 earlystop=tf.keras.callbacks.EarlyStopping(patience=5)
15 callbacks=[earlystop]
16 history=model.fit(x=X[train], y=Y[train], validation_split
17                  =0.2, epochs=50, batch_size=128, shuffle=True, callbacks=
18                  callbacks)
```

Listing 14: Aggiunta di due layer densi nella prima versione del secondo modello

Anche per questo codice valgono le considerazioni che descritte nel primo modello (vedere capitolo 4.3.1.1), con le due uniche eccezioni:

- da Tensorflow a Keras: il modello VGGFace non è supportato da Tensorflow, per tale motivazione è stato necessario caricare i layer usufruendo esclusivamente di Keras
- *last\_layer*: gli strati della VGG-16 pre-allenata su VGGFace hanno un nome differente dal precedente modello, quindi anche l'ultimo strato deve essere richiamato con il nome presente nella rete (mostrato in figura 37)

Essendo infatti un modello molto simile al primo analizzato, gode degli stessi problemi: è presente un leggero *underfitting* dell'architettura complessiva, ad indice del fatto che la versione appena descritta sia una rete troppo semplice per il task in questione, e con pochi parametri. Questa ragione è servita per implementare una seconda versione, con lo scopo di aumentare i parametri della rete stessa e sanare il lieve *underfitting*.

#### 4.3.2.2 Pain\_detection\_model2\_v2

Nella seconda versione del modello VGG-16 pre-addestrato su VGGFace, dunque, abbiamo aumentato il grado di complessità addestrando più layer dell'architettura stessa. La metodologia di aggiunta di parametri alla rete però, non è stata la stessa del precedente modello, bensì si è pensato di impostare alcuni layer della VGG-16 come “addestrabili”, al posto di aggiungere in testa due strati convoluzionali.

Quindi, riprendendo la struttura della rete pre-addestrata, ovvero quella mostrata in figure 37, nella versione 2 del presente modello siamo andati a “scongellare” gli ultimi due layer, rendendo i layer:

- *pool5*
- *conv5\_3*

come addestrabili. Per far ciò abbiamo modificato il codice *lst:free* inserendo un ciclo *for* per impostare il numero di strati complessivamente addestrabili, come mostrato nel codice seguente.

```
1 image_size = 224
2 input_shape = (image_size, image_size, 3)
3
4 pre_trained_model= VGGFace(input_shape=input_shape, include_top
   =False)
5 for layer in pre_trained_model.layers:
6     layer.trainable = False
7
8 for layer in pre_trained_model.layers[17:]:
9     layer.trainable = True
10
11 for layer in pre_trained_model.layers:
12     print(layer, layer.trainable)
13
14 pre_trained_model.summary()
```

Listing 15: Unfreezing degli ultimi due layer della VGG-16

Come si può osservare dal codice 15, l'unica differenza è il secondo ciclo *for*, dove sono stati impostati i due strati finali allenabili, con un parametro *layer.trainable=True*.

Dunque, per far chiarezza mostriamo il risultato dell'operazione di *printing* per la presente versione, in figura 38.

```

<keras.engine.input_layer.InputLayer object at 0x7f3941c4fd10> False
<keras.layers.convolutional.Conv2D object at 0x7f3941a746d0> False
<keras.layers.convolutional.Conv2D object at 0x7f3941c1fd50> False
<keras.layers.pooling.MaxPooling2D object at 0x7f3941a74d90> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b1bbd0> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b21090> False
<keras.layers.pooling.MaxPooling2D object at 0x7f3941b18e90> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b28710> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b2ec50> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b35f50> False
<keras.layers.pooling.MaxPooling2D object at 0x7f3941b28c10> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b388d0> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b43110> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b3d810> False
<keras.layers.pooling.MaxPooling2D object at 0x7f3941b4e6d0> False
<keras.layers.convolutional.Conv2D object at 0x7f3941ad0a10> False
<keras.layers.convolutional.Conv2D object at 0x7f3941ad5fd0> False
<keras.layers.convolutional.Conv2D object at 0x7f3941b49bd0> True
<keras.layers.pooling.MaxPooling2D object at 0x7f3941ae42d0> True
Model: "vggface_vgg16"

```

Figura 38: Riassunto degli strati allenabili e non allenabili

Avendo aumentato solo il numero di parametri, senza modificare l'architettura di rete quindi senza aggiungere due strati convoluzionali, ma solo “scongelando” gli ultimi layer, gli script di addestramento risultano uguali a quelli descritti nella precedente versione (vedere codice 14).

Come vedremo successivamente, con questa seconda versione il problema dell'*underfitting* viene sanato, dato che il modello è stato reso più complesso e più ricco di parametri.

### 4.3.3 Terzo modello: rete ResNet-50 pre-addestrata su dataset di volti

L'ultimo modello che abbiamo testato, si basa su un'altra tipologia di rete, completamente diversa dalla VGG-16: la ResNet-50 descritta nel capitolo 4.3. Tale architettura, come accennato precedentemente, ha un totale di 50 strati e risulta pre-addestrata sulla seconda versione del database VGGface, denominata VGGFace2. Gli step iniziali implementati per la costruzione del presente modello risultano uguali ai passi descritti nel capitolo 4.3.1, ovvero:

- Conversione delle immagini in *numpy arrays* come riportato nel codice 4
- Etichettatura delle immagini con label binarie (0 ed 1), come mostrato nel codice 5

- Importazione delle librerie di interesse, comprensivo della libreria *vgg-face*, come da codice 11

Successivamente a questi step, seguendo la pipeline riportata anche negli altri modelli, abbiamo definito la tipologia di rete pre-addestrata di interesse e abbiamo caricato i pesi specificando il dataset, mediante il seguente codice:

```
1 image_size = 224
2 input_shape = (image_size, image_size, 3)
3
4 pre_trained_model=VGGFace(model='resnet50',input_shape=
   input_shape, include_top=False)
```

Listing 16: Caricamento della rete ResNet-50 con relativi pesi

Come si può notare dal codice, in questo caso è stato specificato un modello tramite il parametro *model*, rispetto al codice analizzato precedentemente (12); questo perchè specificando il tipo di rete da prendere in considerazione, si modifica anche il dataset da cui caricare i pesi (in questo caso il database VGGFace2). Quindi, se il parametro *model* non viene inserito, di default la rete che viene presa in considerazione è la VGG-16 allenata su VGGFace, altrimenti se specificato si utilizza l'architettura voluta con il database relativo.

Del presente modello è stata testata esclusivamente una versione, in quanto le performance di rete risultavano già soddisfacenti e prive di *overfitting/undefitting*, presenti nei precedenti training.

#### 4.3.3.1 Pain\_detection\_model3.v1

Come le altre prime versioni precedenti a tale modello, anche qui si è deciso di andare ad impostare tutti i layer della ResNet-50 come *non addestrabili*, per minimizzare il numero di parametri da allenare.

```
1 for layer in pre_trained_model.layers:
2     layer.trainable = False
3
4 for layer in pre_trained_model.layers:
5     print(layer, layer.trainable)
6
7 pre_trained_model.summary()
```

Listing 17: Layer non allenabili sulla rete ResNet-50

Di seguito mostriamo il risultato del *print* <sup>10</sup>.

<sup>10</sup> Della struttura complessiva, sono stati presi esclusivamente gli strati finali, per mostrare le parti salienti dell'architettura stessa

```

<keras.layers.core.Activation object at 0x7f53b6062f10> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7c0b50> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e7cb4d0> False
<keras.layers.merge.Add object at 0x7f539e7ce850> False
<keras.layers.core.Activation object at 0x7f539e7d55d0> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7c4a10> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f53b6062110> False
<keras.layers.core.Activation object at 0x7f53b6043d90> False
<keras.layers.convolutional.Conv2D object at 0x7f53b6049d50> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f53b6023e10> False
<keras.layers.core.Activation object at 0x7f53b608e150> False
<keras.layers.convolutional.Conv2D object at 0x7f53b608a710> False
<keras.layers.convolutional.Conv2D object at 0x7f53b606dd90> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f53b603e190> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f53b60d7050> False
<keras.layers.merge.Add object at 0x7f53b61328d0> False
<keras.layers.core.Activation object at 0x7f53b60dfd50> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7de2d0> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f53b60c9bd0> False
<keras.layers.core.Activation object at 0x7f53b60f8210> False
<keras.layers.convolutional.Conv2D object at 0x7f53b614ec10> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e7eb950> False
<keras.layers.core.Activation object at 0x7f539e7e9910> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7e6d90> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e7f3ed0> False
<keras.layers.merge.Add object at 0x7f53b6116690> False
<keras.layers.core.Activation object at 0x7f539e7f31d0> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7864d0> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e789890> False
<keras.layers.core.Activation object at 0x7f539e789550> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7913d0> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e7957d0> False
<keras.layers.core.Activation object at 0x7f539e7a06d0> False
<keras.layers.convolutional.Conv2D object at 0x7f539e7f34d0> False
<keras.layers.normalization.batch_normalization.BatchNormalization object at 0x7f539e7aa710> False
<keras.layers.merge.Add object at 0x7f539e7f33d0> False
<keras.layers.core.Activation object at 0x7f539e7b2990> False
<keras.layers.pooling.AveragePooling2D object at 0x7f539e73d1d0> False
Model: "vggface_resnet50"

```

Figura 39: Riassunto degli ultimi strati allenabili e non allenabili della ResNet-50



conv5_1_1x1_proj/bn (BatchNorma	(None, 7, 7, 2048)	8192	conv5_1_1x1_proj[0][0]
add_13 (Add)	(None, 7, 7, 2048)	0	conv5_1_1x1_increase/bn[0][0] conv5_1_1x1_proj/bn[0][0]
activation_42 (Activation)	(None, 7, 7, 2048)	0	add_13[0][0]
conv5_2_1x1_reduce (Conv2D)	(None, 7, 7, 512)	1048576	activation_42[0][0]
conv5_2_1x1_reduce/bn (BatchNor	(None, 7, 7, 512)	2048	conv5_2_1x1_reduce[0][0]
activation_43 (Activation)	(None, 7, 7, 512)	0	conv5_2_1x1_reduce/bn[0][0]
conv5_2_3x3 (Conv2D)	(None, 7, 7, 512)	2359296	activation_43[0][0]
conv5_2_3x3/bn (BatchNormalizat	(None, 7, 7, 512)	2048	conv5_2_3x3[0][0]
activation_44 (Activation)	(None, 7, 7, 512)	0	conv5_2_3x3/bn[0][0]
conv5_2_1x1_increase (Conv2D)	(None, 7, 7, 2048)	1048576	activation_44[0][0]
conv5_2_1x1_increase/bn (BatchN	(None, 7, 7, 2048)	8192	conv5_2_1x1_increase[0][0]
add_14 (Add)	(None, 7, 7, 2048)	0	conv5_2_1x1_increase/bn[0][0] activation_42[0][0]
activation_45 (Activation)	(None, 7, 7, 2048)	0	add_14[0][0]
conv5_3_1x1_reduce (Conv2D)	(None, 7, 7, 512)	1048576	activation_45[0][0]
conv5_3_1x1_reduce/bn (BatchNor	(None, 7, 7, 512)	2048	conv5_3_1x1_reduce[0][0]
activation_46 (Activation)	(None, 7, 7, 512)	0	conv5_3_1x1_reduce/bn[0][0]
conv5_3_3x3 (Conv2D)	(None, 7, 7, 512)	2359296	activation_46[0][0]
conv5_3_3x3/bn (BatchNormalizat	(None, 7, 7, 512)	2048	conv5_3_3x3[0][0]
activation_47 (Activation)	(None, 7, 7, 512)	0	conv5_3_3x3/bn[0][0]
conv5_3_1x1_increase (Conv2D)	(None, 7, 7, 2048)	1048576	activation_47[0][0]
conv5_3_1x1_increase/bn (BatchN	(None, 7, 7, 2048)	8192	conv5_3_1x1_increase[0][0]
add_15 (Add)	(None, 7, 7, 2048)	0	conv5_3_1x1_increase/bn[0][0] activation_45[0][0]
activation_48 (Activation)	(None, 7, 7, 2048)	0	add_15[0][0]
avg_pool (AveragePooling2D)	(None, 1, 1, 2048)	0	activation_48[0][0]
=====			
Total params: 23,561,152			
Trainable params: 0			
Non-trainable params: 23,561,152			

Figura 40: Strati finali della struttura complessiva

Come si può osservare dalla figura 40, vi è una quarta colonna aggiuntiva rispetto agli altri modelli: il parametro *connected\_to*, che esprime semplicemente il layer precedente a cui il layer corrente risulta connesso.

Partendo da questa impostazione, ovvero “congelando” tutti gli strati della ResNet-50 deputati all’estrazione delle features, abbiamo proceduto aggiungendo due layer *Dense* in testa alla struttura, come già discusso anche nelle prime versioni precedenti. Il codice seguente mostra la metodologia di aggiunta di due strati densi in testa alla rete, intervallati sempre da uno

strato di Dropout con rate al 50%.

```
1 last_layer = pre_trained_model.get_layer('avg_pool')
2 last_output = last_layer.output
3 x = GlobalMaxPooling2D()(last_output)
4 x = Dense(512, activation='relu')(x)
5 x = Dropout(0.5)(x)
6 x = Dense(2, activation='softmax')(x)
7
8 model = Model(pre_trained_model.input, x)
9 model.compile(loss='categorical_crossentropy',
10               optimizer=tf.keras.optimizers.SGD(learning_rate
11               =0.001, momentum=0.9),
12               metrics=['accuracy'])
13
14 earllystop=tf.keras.callbacks.EarlyStopping(patience=5)
15 callbacks=[earllystop]
16 history=model.fit(x=X[train], y=Y[train], validation_split
17                  =0.2, epochs=50, batch_size=128, shuffle=True, callbacks=
18                  callbacks)
```

Listing 18: Layer non allenabili sulla rete ResNet-50

Il codice risulta molto simile a quello già analizzato nella precedente versione della VGG-16 allenata con VGGface ovvero il Listing 14. Anche qui, come si nota è stato necessario specificare i diversi strati tramite l'utilizzo esclusivo della libreria *Keras*, inoltre è stato modificato il parametro *last\_layer* impostando l'ultimo layer della ResNet-50.

Come accennato precedentemente, non è stato necessario implementare una seconda versione, poichè il modello risultava privo di problemi e con delle buone metriche, come discuteremo in dettaglio nel prossimo capitolo.

## 5 Risultati

In questa nuova sezione discuteremo i risultati ottenuti per ogni tipo di modello discusso nel capitolo precedente. In particolare affronteremo i seguenti argomenti:

- Metriche di classificazione: parleremo in dettaglio mostrando anche formule matematiche relative agli indici di prestazione analizzati;
- Performance del modello: discuteremo le prestazioni dei modelli analizzati mostrando per ognuno di essi il valore dei diversi indici di bontà.

## 5.1 Metriche di classificazione

I modelli di ML forniscono sempre delle previsioni che hanno un carattere probabilistico. Anche quando riportiamo dei risultati in forma assoluta (es: la diagnosi fornita dal modello è positiva, vi è la malattia) dobbiamo ricordarci che, dietro le quinte, il modello ha fornito una probabilità superiore ad una soglia che abbiamo fissato, e, in base a questa soglia, abbiamo deciso che il modello ha detto dato un esito positivo.

Ovviamente, non possiamo attenderci che il modello produca sempre risultati corretti. Anche se ci limitiamo alla valutazione su un insieme di dati di test, in generale su molti di questi dati il modello fornirà una previsione corretta, ma su alcuni, si spera pochi, produrrà una previsione sbagliata.

E' importante dunque, misurare le prestazioni del modello tramite degli indicatori che ne sappiano descrivere la sua bontà.

Ma esistono differenti metriche utilizzabili per quantificare le performance di un modello. Alcune sono più “a grana larga”, altre consentono di tenere sotto controllo alcuni aspetti di maggior dettaglio.

Esaminiamo le varie metriche dandone la definizione, chiarendone l'utilità e gli eventuali limiti. Le metriche prese in considerazione saranno le seguenti:

- Matrice di confusione
- Precision
- Recall
- F1-score
- Specificità
- Roc-curve (AUC)

**Matrice di confusione** : Nel campo del machine learning e nella classificazione statistica, una matrice di confusione è una tabella in cui le previsioni sono rappresentate nelle colonne e lo stato effettivo è rappresentato dalle righe (a volte questo è invertito, con istanze reali in colonne e previsioni nelle righe).

La tabella è un'estensione della matrice di confusione nell'analisi predittiva e rende facile vedere se si è verificato un errore di denominazione e se le previsioni sono più o meno corrette.

Da questa tabella, quindi, è possibile comprendere le performance di un modello predittivo di classificazione in modo da determinare quanto questo modello sia accurato ed efficace.

Per una maggior chiarezza sul funzionamento della matrice di confusione, introduciamo un esempio, ipotizzando una classificazione binaria (caso di nostro interesse).

Un classificatore binario produce output con due valori di classe o etichette, come Si / No e 1/0, per dati di input conosciuti. La classe di interesse viene solitamente indicata come “positiva” e l’altra come “negativa“. Immaginiamo che sulla base di dati tratti da prelievi istologici il nostro modello debba prevedere se un campione di tessuto proviene da un tumore benigno o maligno (è il caso del Wisconsin Breast Cancer Dataset <sup>11</sup>). La matrice di confusione consente di rappresentare un livello di dettaglio maggiore della sola accuratezza, quantificando separatamente i falsi positivi ed i falsi negativi, e risulta così composta:

Confusion Matrix		
	Valori	predetti
Valori	<b>TP</b>	<b>FN</b>
reali	<b>FP</b>	<b>TN</b>

Figura 41: Struttura generale della matrice di confusione per classificazione binaria

Per comprendere meglio l’incrocio tra valori effettivi e previsti alla matrice di confusione vengono associati i seguenti termini:

- True positive (TP, o veri positivi): sono i casi in cui abbiamo previsto che la mail fosse spam, e lo sono realmente

---

<sup>11</sup> [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

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

Facendo riferimento alla precedente lista possiamo descrivere tutte le metriche di base utilizzate nel presente lavoro.

**Precision (o precisione)** : La precisione è l'abilità di un classificatore di non etichettare un'istanza positiva che è in realtà negativa. Per ogni classe è definito come il rapporto tra veri positivi e la somma di veri e falsi positivi. Detto in un altro modo, “per tutte le istanze classificate come positive, quale percentuale era corretta?”. La formula di calcolo della precisione è la seguente:

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

Figura 42: Formula per il calcolo della metrica di *precisione*

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

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

Figura 43: Formula per il calcolo della metrica di *richiamo*

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

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

Figura 44: Formula per il calcolo della metrica di *F1-score*

**Specificity (o specificità, SP)** : è calcolata come il numero di previsioni negative corrette diviso per il numero totale di negativi (N). Viene anche chiamato vero tasso negativo (TNR). La migliore specificità è 1, mentre la peggiore è 0. Si può usare questa formula per il calcolo:

$$SP = \frac{TN}{TN + FP} = \frac{TN}{N}$$

Figura 45: Formula per il calcolo della metrica di *specificità*

**ROC-curve (AUC)** : la curva ROC viene costruita considerando tutti i possibili valori del test e, per ognuno di questi, si calcola la proporzione di veri positivi (la sensibilità) e la proporzione di falsi positivi. La proporzione di falsi positivi si calcola con la formula standard: 1 - specificità. Congiungendo i punti che mettono in rapporto la proporzione di veri positivi e di falsi positivi (le cosiddette coordinate) si ottiene una curva chiamata *curva ROC*. L'area sottostante alla curva ROC (AUC, acronimo dei termini inglesi "Area Under the Curve") è una misura di accuratezza diagnostica. Se un ipotetico nuovo test discriminasse perfettamente tra le classi prese in esame, l'area della curva ROC avrebbe valore 1, cioè il 100% di accuratezza. Nel caso in cui il nuovo test non discriminasse per niente tra le classi, la curva ROC avrebbe un'area di 0.5 (o 50%) che coinciderebbe con l'area sottostante la diagonale del grafico (Fig. 46).

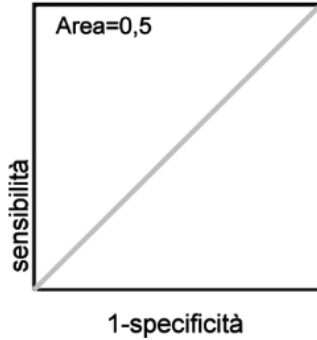


Figura 46: Rappresentazione della metrica  $AUC$  con un valore di 0.5

Nella realtà, si considera adeguato un test diagnostico con un'area sotto la curva  $\geq 80\%$ . L'area sotto la curva può assumere valori compresi tra 0.5 e 1.0. Tanto maggiore è l'area sotto la curva (cioè tanto più la curva si avvicina al vertice del grafico) tanto maggiore è il potere discriminante del test. Per l'interpretazione dei valori dell'area sottostante la curva ROC è possibile riferirsi alla classificazione proposta da Swets [12]:

- $AUC=0.5$  il test non è informativo;
- $0.5 \leq AUC \leq 0.7$  il test è poco accurato;
- $0.7 \leq AUC \leq 0.9$  il test è moderatamente accurato;
- $0.9 \leq AUC \leq 1.0$  il test è altamente accurato;
- $AUC=1$  test perfetto.

Definite le metriche di modello prese in considerazione, nel prossimo capitolo discuteremo in dettaglio le performance ottenute da ogni architettura testata.

## 5.2 Performance del modello

Nel presente capitolo andremo a mostrare i risultati ottenuti dalle diverse versioni dei modelli discussi in precedenza. Le performance saranno mostrate secondo la seguente modalità:

- Tabella riassuntiva: una tabella che riporta tutti gli indici di bontà discussi nel capitolo 5.1, riportando sia le performance per ogni split, sia le metriche medie considerando le suddivisioni complessivamente;

- Roc-curve: per ogni versione sarà mostrata anche la curva Roc in cui viene raffigurata la metrica AUC sia per ogni split, sia come medis complessiva.

### 5.2.1 Primo modello: rete VGG-16 pre-addestrata su dataset ImageNet

Come primo modello facciamo riferimento a quanto descritto nel capitolo 4.3.1.

Ricordiamo che questo modello testa la rete VGG-16 pre-allenata su ImageNet, in cui abbiamo applicato la tecnica del *fine tuning*, dapprima senza aggiunta di complessità, e nella seconda versione aggiungendo due layer *Conv* per aumentare il numero di parametri e rendere l'architettura complessiva più complessa.

#### 5.2.1.1 Pain\_detection\_model1\_v1

Le performance di modello per la prima versione del presente modello risultano raffigurate nella tabella 47 e in figura 48.

Metriche considerate	Split 1	Split 2	Split 3	Split 4	Split 5
Losses	0.11292986	0.09558747	0.10336502	0.08903932	0.09316113
Accuracies	0.9555819	0.96801895	0.96742672	0.9671306	0.9715724
Sensitivities	0.96515246	0.97137523	0.95644057	0.97759801	0.98568762
Specificities	0.94689266	0.96497175	0.97740113	0.95762712	0.95875706
F1-scores	0.95387454	0.96656347	0.96545226	0.96587765	0.97058824
Avg loss	0.09881635904312133 +/- 0.008460479618981083				
Avg accuracy	0.9659461140632629 +/- 0.005421508760434097				
Avg sensitivity	0.9712507778469197 +/- 0.010052430868106385				
Avg specificity	0.9611299435028249 +/- 0.010001117113382509				
Avg F1-score	0.9644712308474015 +/- 0.005604025706811158				

Figura 47: Metriche complessive medie



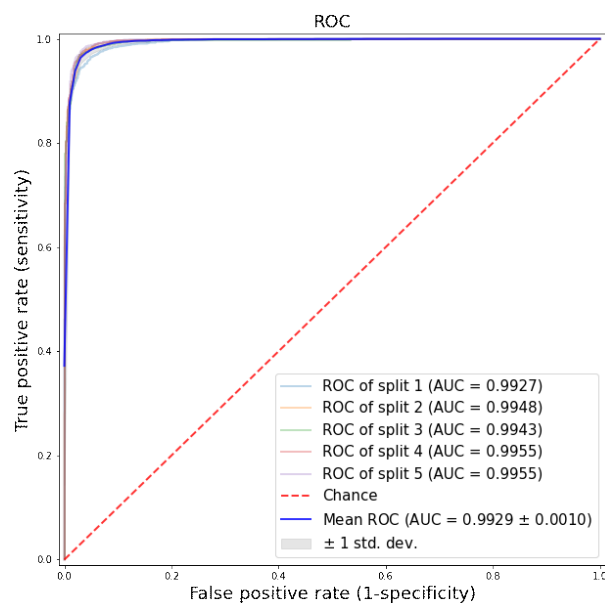


Figura 48: Curva ROC con relativa metrica AUC media

Come si nota dall'immagine le metriche sono già sufficientemente alte. In questa prima versione, tuttavia, è presente il problema dell'*underfitting* come discusso in precedenza.

Dunque anche se le performance risultano sufficienti è stato necessario sanare tale problema.

#### 5.2.1.2 Pain\_detection\_model1\_v2

Le performance della seconda versione , costruita per sanare il problema dell'*underfitting*, risultano mostrate in figura 49 e 50.

Mettriche considerate	Split 1	Split 2	Split 3	Split 4	Split 5
Losses	0.11292986	0.09558747	0.10336502	0.08903932	0.09316113
Accuracies	0.9555819	0.96801895	0.96742672	0.9671306	0.9715724
Sensitivities	0.96515246	0.97137523	0.95644057	0.97759801	0.98568762
Specificities	0.94689266	0.96497175	0.97740113	0.95762712	0.95875706
F1-scores	0.95387454	0.96656347	0.96545226	0.96587765	0.97058824
Avg loss	0.09881635904312133 +/- 0.008460479618981083				
Avg accuracy	0.9659461140632629 +/- 0.005421508760434097				
Avg sensitivity	0.9712507778469197 +/- 0.010052430868106385				
Avg specificity	0.9611299435028249 +/- 0.010001117113382509				
Avg F1-score	0.9644712308474015 +/- 0.005604025706811158				

Figura 49: Mettriche compressive medie

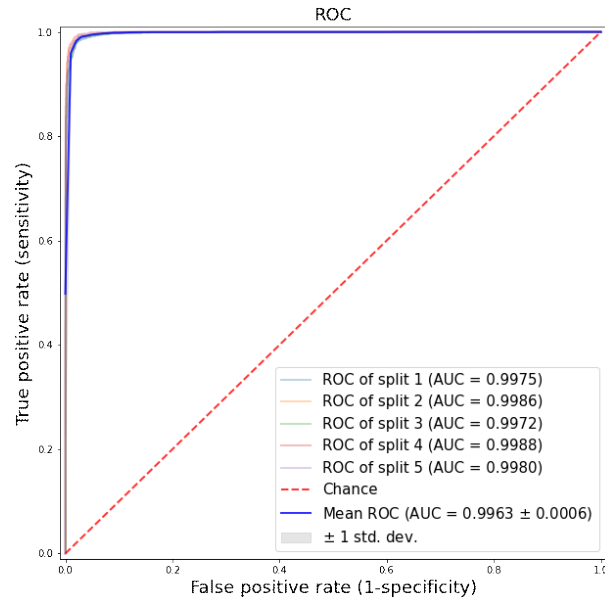


Figura 50: Curva ROC con relativa metrica AUC media

Come si può osservare dalla figura, non solo le metriche sono lievemente aumentate rispetto alla prima versione, ma in generale il modello risulta con performance più veritiere.

### 5.2.2 Secondo modello: rete VGG-16 pre-addestrata su dataset di volti

In questo capitolo illustreremo le metriche relativi al secondo modello testato, spiegato in dettaglio nel capitolo 4.3.2. Ricordiamo che nella presente architettura abbiamo preso in considerazione la rete VGG-16 pre-addestrata su VGGface. Anche qui abbiamo utilizzato la tecnica del *fine tuning*, implementando due versioni, la prima senza complessità e la seconda con l'aggiunta di parametri.

#### 5.2.2.1 Pain\_detection\_model2\_v1

In questa sezione mostriamo i risultati della prima versione del presente modello. Le performance sono schematizzate nella tabella 51 ed in figura 52.

Metriche considerate	Split 1	Split 2	Split 3	Split 4	Split 5
Losses	0.10624789	0.10624789	0.12281611	0.14448629	0.12552017
Accuracies	0.96239263	0.96742672	0.95084393	0.94225645	0.94936335
Sensitivities	0.9639079	0.96017424	0.95457374	0.92159303	0.95706285
Specificities	0.96101695	0.9740113	0.94745763	0.96101695	0.94237288
F1-scores	0.96062016	0.96558198	0.94867038	0.9382325	0.947336
Avg loss	0.11745485663414001 +/- 0.019003735803461257				
Avg accuracy	0.9544566154479981 +/- 0.009154678578053504				
Avg sensitivity	0.9514623522090853 +/- 0.015257859421308686				
Avg specificity	0.9571751412429379 +/- 0.011193857040502143				
Avg F1-score	0.9520882022848653 +/- 0.009809399833901224				

Figura 51: Metriche compressive medie

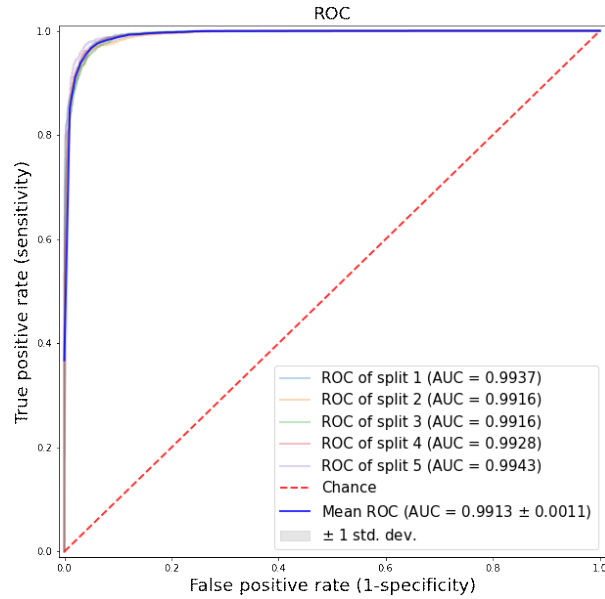


Figura 52: Curva ROC con relativa metrica AUC media

Le metriche raffigurate della seconda versione, risultano più basse del precedente modello, in più anche nella presente architettura vi è il problema dell'*underfitting* dato dall'eccessiva semplicità del modello.

#### 5.2.2.2 Pain\_detection\_model2\_v2

Nella seconda versione del secondo modello abbiamo aumentato la complessità, “scongelando” gli ultimi layer della rete e superando il problema sopra citato. In questo caso le metriche sono mostrate in figura 53 e 54.

Metrische considerate	Split 1	Split 2	Split 3	Split 4	Split 5
Losses	0.05494194	0.06927551	0.07746041	0.04257987	0.0582073
Accuracies	0.98252887	0.97571808	0.97482973	0.98726678	0.98489785
Sensitivities	0.97448662	0.96764157	0.97137523	0.99751089	0.99128811
Specificities	0.98983051	0.98305085	0.9779661	0.9779661	0.97909605
F1-scores	0.9815105	0.97431078	0.97349548	0.98676516	0.98424467
Avg loss	0.060493004322052 +/- 0.012016193717211694				
Avg accuracy	0.9810482621192932 +/- 0.0049550461109135685				
Avg sensitivity	0.9804604853764779 +/- 0.011751658129795036				
Avg specificity	0.9815819209039548 +/- 0.004528240657517281				
Avg F1-score	0.9800653166722404 +/- 0.005305109467633541				

Figura 53: Metrische compressive medie

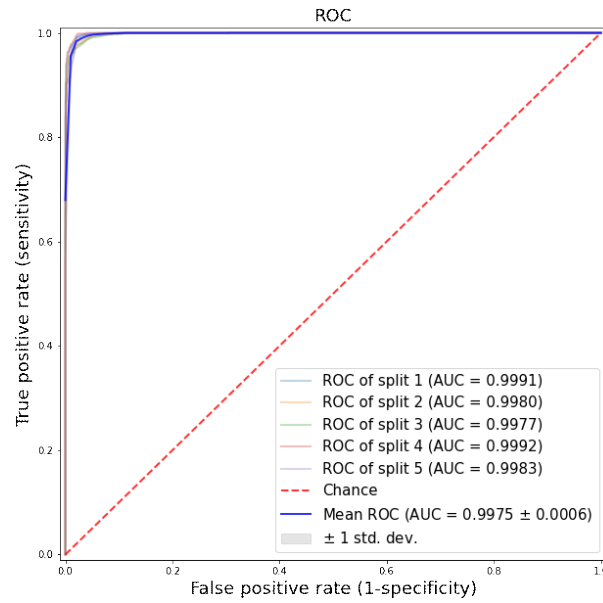


Figura 54: Curva ROC con relativa metrica AUC media

Le performance risultano le più alte ottenute in questo lavoro, superando il problema dell'*underfitting* e raggiungendo un totale del 98% in tutte le metriche considerate, arrivando infine ad un AUC medio pari al 99,75%.

### 5.2.3 Terzo modello: rete ResNet-50 pre-addestrata su dataset di volti

L'ultimo modello testato, descritto in dettaglio nel capitolo 4.3.3 infine, prende in esame la rete ResNet-50 pre-addestrata su VGGFace. In questo caso è stato necessario implementare una singola versione, essendo il modello già sufficientemente complesso.

#### 5.2.3.1 Pain\_detection\_model3.v1

In questa prima versione del terzo modello abbiamo mantenuto i pesi estratti dal pre-addestramento, senza aggiunta di ulteriore complessità. Le metriche sono mostrate in tabella 55 ed in figura 56.

Metriche considerate	Split 1	Split 2	Split 3	Split 4	Split 5
Losses	0.05770753	0.05093358	0.05630437	0.0508533	0.0539105
Accuracies	0.97571808	0.98223275	0.9786793	0.97897542	0.98015988
Sensitivities	0.97635345	0.98568762	0.98319851	0.97946484	0.98568762
Specificities	0.97514124	0.97909605	0.97457627	0.97853107	0.97514124
F1-scores	0.97453416	0.98141264	0.97772277	0.97794346	0.97928903
Avg loss	0.05394185557961464 +/- 0.0027695353600160864				
Avg accuracy	0.9791530847549439 +/- 0.002123829498717367				
Avg sensitivity	0.9820784069695083 +/- 0.0036582298585436514				
Avg specificity	0.9764971751412428 +/- 0.00191090785596473				
Avg F1-score	0.9781804120300626 +/- 0.0022466438266927485				

Figura 55: Metriche compressive medie

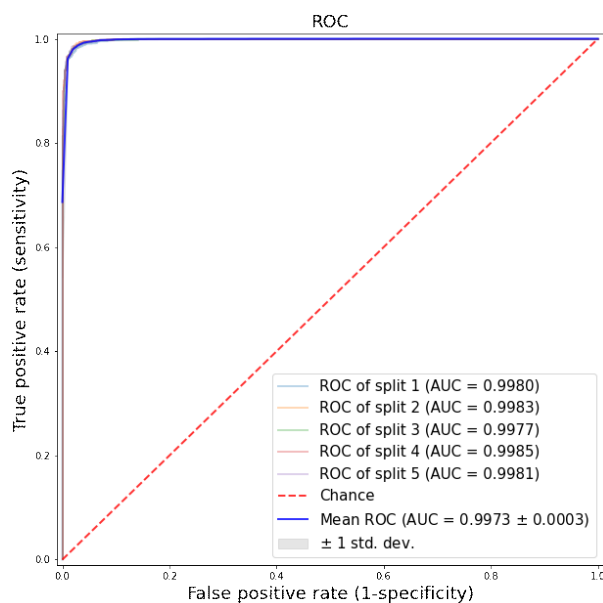


Figura 56: Curva ROC con relativa metrica AUC media

Complessivamente le metriche risultano discretamente alte, tuttavia sono lievemente più basse delle performance della versione precedente (Fig. 53 e 54).

## 6 Discussione dei risultati e conclusione

In quest'ultimo capitolo faremo delle considerazioni finali sul lavoro descritto, discutendo i risultati ottenuti.

Riassuntivamente, nel presente lavoro sono stati sfruttati tre modelli pre-addestrati:

- VGG-16 su ImageNet
- VGG-16 su VGGFace
- ResNet-50 su VGGFace2

1. **Pain\_detection\_model1\_v1**: Il modello pre-addestrato (VGG-16) è costituito da 19 strati “congelati” deputati all'estrazione delle features (come mostrato in Fig. 33 e 34), ai quali abbiamo aggiunto in cascata

1 strato piatto, 1 strato densamente connesso con funzione di attivazione *ReLU*, 1 strato denso e 1 ultimo strato densamente connesso con funzione di attivazione *SoftMax*, deputati alla classificazione. I pesi sono stati caricati da ImageNet, ed i risultati ottenuti sono riportati in figura 47 e 48. Il modello presenta un leggerissimo under-fitting, per sanare il quale avremmo potuto:

- Scongellare alcuni strati convoluzionali preesistenti al fine di aumentare il numero di parametri addestrabili;
- Aumentare la complessità della rete mediante l'aggiunta di ulteriori strati convoluzionali.

2. **Pain\_detection\_model1\_v2:** Il modello pre-addestrato (VGG-16) è costituito da 19 strati “congelati” deputati all'estrazione delle features (come mostrato in figura 33 e 34), ai quali abbiamo aggiunto in cascata altri 2 strati convoluzionali, 1 strato piatto, 1 strato densamente connesso con funzione di attivazione *ReLU*, 1 strato denso e 1 ultimo strato densamente connesso con funzione di attivazione *SoftMax*. I pesi sono stati caricati da ImageNet, ed i risultati sono mostrati in tabella 49 ed in figura 50. L'aggiunta di complessità al modello ha sanato l'under-fitting della prima versione.
3. **Pain\_detection\_model2\_v1:** Il modello pre-addestrato (VGG-16) è costituito da 19 strati “congelati” deputati all'estrazione delle features (rappresentati figura 36 e 37), ai quali abbiamo aggiunto in cascata 1 strato piatto, 1 strato densamente connesso con funzione di attivazione *ReLU*, 1 strato denso e 1 ultimo strato densamente connesso con funzione di attivazione *SoftMax*, deputati alla classificazione. I pesi sono stati caricati da VGGFace, e le metriche sono mostrate in figura 51 e 52. Il modello presenta un leggerissimo under-fitting. Per risolverlo in questo caso abbiamo aumentato la complessità della rete mediante lo “scongellamento” di solo due strati della rete stessa (versione 2).
4. **Pain\_detection\_model2\_v2:** Il modello pre-addestrato (VGG-16) è costituito da 17 strati “congelati” deputati all'estrazione delle features (come mostra la figura 38), ai quali abbiamo aggiunto in cascata 1 strato piatto, 1 strato densamente connesso con funzione di attivazione *ReLU*, 1 strato denso e 1 ultimo strato densamente connesso con funzione di attivazione *SoftMax*. Anche qui i pesi sono stati caricati da VGGFace, e le metriche sono mostrate nella tabella 53 ed in figura 54.



Il modello con aggiunta di complessità sana l'under-fitting restituendo le metriche più alte ottenute nel presente progetto.

5. **Pain\_detection\_model3\_v1:** Il modello pre-addestrato (ResNet50) è costituito da 194 strati “congelati” deputati all'estrazione delle features (rappresentati in figura 39 e 40), ai quali abbiamo aggiunto in cascata 1 strato piatto, 1 strato densamente connesso con funzione di attivazione *ReLU*, 1 strato denso e 1 ultimo strato densamente connesso con funzione di attivazione *SoftMax*, deputati alla classificazione. I pesi sono stati caricati da VGGFace2, e le performance di modello sono raffigurate nella tabella 55 ed in figura 56. Il modello non presenta né over- né under-fitting, di conseguenza non è stato necessario intervenire per sanare i medesimi.

Inoltre abbiamo comparato il lavoro da noi proposto con gli articoli presenti in letteratura (riportati nel capitolo 3.3) concendrandoci esclusivamente sui paper che fossero simili al progetto qui presente, ossia selezionando gli articoli in base :

- ad una classificazione compiuta tramite PSPI
- al database “*UNBC-McMaster Shoulder Pain Achieve database*”

Tale confronto è rappresentato in tabella 57 e 58.

Paper	Publication year	Dataset	Pre-processing	Pre-processed image resolution	Features extraction and classification	PSPI-based classification	Classification metrics(ACC,SEN, SPE,F1,AUC)
Enhanced Deep Learning Algorithm Development to detect pain intensity from facial expression images – Ghazal Bargshady et al.	2020	UNBC-McMaster Shoulder Pain Achieve database	<ul style="list-style-type: none"> <li>o Normalization</li> <li>o Cropping</li> <li>o Centralizing</li> <li>o Resizing</li> </ul>	224x224x3	1. Pre-trained VGGFace (feature extraction) 2. PCA (dimensionality reduction) 3. Enhanced joint hybrid CNN-BiLSTM (classification)	PSPI: - 0=No Pain - 1=Weak pain - 2,3=Mid pain - 4= Strong pain	10-fold cross val: - <b>ACC:</b> 90%(testset) - <b>AUC:</b> 98.4%
Spatio-temporal Pain Recognition in CNN-Based Super-Resolved Facial Images – Bellantonì Marco et al.	2016	UNBC-McMaster Shoulder Pain Achieve database	<ul style="list-style-type: none"> <li>o Variazione di risoluzione, tramite algoritmi SR o down-up sampling</li> <li>o Generazione di immagini SR tramite l'algoritmo "example-based learning"</li> </ul>	224x224x3	1. Pre-trained VGGFace (feature extraction) 2. LSTM (classification)	PSPI: - <1=No pain - Da 2 a 6= weak pain - 6= Strong Pain	<b>ACC Per Dataset:</b> - Dataset1: 63.43% - Dataset2: 62.64% - Dataset3: 65.34% <b>F1-SCORE per Dataset:</b> - Dataset1: 67% - Dataset2: 66% - Dataset3: 69%

Figura 57: Prima tabella di confronto del lavoro proposto con articoli in letteratura

Pain Recognition and Intensity Classification Using Facial Expressions - W. A. Shier et al.	2016	UNBC-McMaster Shoulder Pain Achieve database	<ul style="list-style-type: none"> <li>Normalization (Viola-Jones object detection and eye-normalization)</li> </ul>	240x320	1. Gabor energy filter (feature extraction) 2. One-against-one SVM (classification)	PSPI: <ul style="list-style-type: none"> <li>0=No pain</li> <li>&lt; di 3= Weak pain</li> <li>&gt; di 3= Strong Pain</li> </ul>	Pain detection (10 fold cross validation): <ul style="list-style-type: none"> <li><b>Precision :</b> 74%</li> </ul> Pain intensity (10 fold cross validation):  <b>Precision Rate</b> <ul style="list-style-type: none"> <li>No pain: 74%</li> <li>Weak pain: 30%</li> <li>Strong pain: 78%</li> </ul>
<b>Proposed</b>	2021	UNBC-McMaster Shoulder Pain Achieve database	<ul style="list-style-type: none"> <li>Cropping (face detection tramite Viola-Jones)</li> <li>Resizing</li> <li>Normalizing</li> </ul>	224x224x3	<b>Best:</b> Pre-trained VGG-16 on VGGFace + fine tuning (feature extraction and classification)	PSPI: <ul style="list-style-type: none"> <li>0= No Pain</li> <li>&gt; di 0=Pain</li> </ul>	Pain detection (AVG): <ul style="list-style-type: none"> <li><b>ACC</b>= 98.10%</li> <li><b>SEN</b>= 98.04%</li> <li><b>SPE</b>= 98.15%</li> <li><b>F1-score</b>= 98.00%</li> <li><b>AUC</b>= 99.75%</li> </ul>

Figura 58: continuo della tabella di confronto del lavoro proposto con articoli in letteratura

Concludendo, il lavoro descritto nella presente tesi ha proposto una confronto fra diversi modelli pre-addestrati su due tipologie di dataset (immagini naturali e immagini specifiche di volti), sviluppando una classificazione binaria su base PSPI, raggiungendo metriche soddisfacenti che nel caso migliore, ovvero nel modello 4.3.2.2, sono:

- Indici di bontà complessivi: 98%
- AUC: 99%

Il progetto propone anche un'accurata fase di pre-processing dei dati e un'adeguata soluzione per gestire e risolvere il problema dell'underfitting, riscontrato in alcune versioni. Inoltre, da un confronto con recenti lavori analoghi presenti nello stato dell'arte, l'approccio proposto sembra molto promettente al fine di identificare automaticamente il dolore da espressioni del volto. Come sviluppo futuro, provvederemo anche noi ad estendere l'analisi ad una classificazione multi-classe, così da verificare se le ottime performance dell'approccio proposto si rifletteranno anche nell'identificazione di più classi di dolore.

## Riferimenti bibliografici

- [1] Qiong Cao et al. “Vggface2: A dataset for recognising faces across pose and age”. In: *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*. IEEE. 2018, pp. 67–74.
- [2] William P Chapman, Chester M Jones et al. “Variations in cutaneous and visceral pain sensitivity in normal subjects”. In: *The Journal of clinical investigation* 23.1 (1944), pp. 81–91.
- [3] Charles Darwin. “The expression of emotions in man and animals. New York: Philosophical Library”. In: *Original work published* (1872).
- [4] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [5] Paul Ekman e Wallace V Friesen. “Manual of the facial action coding system (FACS)”. In: *Trans. ed. Vol. Consulting Psychologists Press, Palo Alto* (1978).
- [6] Mauro Ercolani e Laura Pasquini. *La percezione del dolore*. Il mulino, 2007.
- [7] Robert J Gatchel et al. “The biopsychosocial approach to chronic pain: scientific advances and future directions.” In: *Psychological bulletin* 133.4 (2007), p. 581.
- [8] Patrick Lucey et al. “Painful data: The UNBC-McMaster shoulder pain expression archive database”. In: *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*. IEEE. 2011, pp. 57–64.
- [9] Akiko Okifuji, Dennis C Turk e Shelly L Curran. “Anger in chronic pain: investigations of anger targets and intensity”. In: *Journal of psychosomatic research* 47.1 (1999), pp. 1–12.
- [10] Omkar M Parkhi, Andrea Vedaldi e Andrew Zisserman. “Deep face recognition”. In: (2015).
- [11] Karen Simonyan e Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
- [12] John A Swets. “Measuring the accuracy of diagnostic systems”. In: *Science* 240.4857 (1988), pp. 1285–1293.
- [13] Dennis C Turk e Ronald Melzack. *Handbook of pain assessment*. Guilford Press, 2011.

- [14] Paul Viola e Michael Jones. “Rapid object detection using a boosted cascade of simple features”. In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. Ieee. 2001, pp. I–I.
- [15] JC Willer, A Roby e D Le Bars. “Psychophysical and electrophysiological approaches to the pain-relieving effects of heterotopic nociceptive stimuli”. In: *Brain* 107.4 (1984), pp. 1095–1112.