Kinship Recognition

Pierluigi Liguori e Fabiano Priore

Progetto di Fondamenti di Visione Artificiale e Biometria
Università degli Studi di Salerno
Dipartimento di Informatica

La *Kinship Recognition* è un' area di ricerca che pone interesse nel verificare l'eventuale esistenza di un legame di parentela tra due persone partendo in genere da un' analisi delle immagini dei volti. I recenti progressi nell' image processing hanno ridestato un coinvolgimento nel settore e nello studio di nuovi algoritmi, i quali trovano applicazione nelle indagini penali, nella sorveglianza e nell' analisi dei Social Network. Inoltre, ulteriori ricerche evidenziano come individui che sono geneticamente correlati, o semplicemente molto somiglianti, tendono ad una maggiore cooperazione nel lavoro di gruppo per via di un alto grado di fiducia che vi si instaura involontariamente; questo effetto è terreno fertile per molteplici applicazioni a livello sociale.
Questo elaborato tratta il riconoscimento di parentela grazie all'ausilio di reti neurali. Nel capitolo introduttivo vengono spiegati nel dettaglio i nostri obiettivi, la strategia risolutiva, i datasets a cui abbiamo fatto riferimento e l'implementazione di una prima soluzione. Nei capitoli successivi vengono condotti diversi esperimenti e nuove soluzioni via via graduali al fine di migliorare l'accuratezza. Dopo aver depositato una *milestone*, vi è un testing su nuovo dateset ottenuto dall'estrazione di volti dei concorrenti delle puntate del programma televisivo dei *"Soliti ignoti"*.

1. INTRODUZIONE

Come dimostrato in (Kinship verification from facial images and videos: human versus machine , 2018), le prestazioni nell'attività di verifica della parentela tra individui sono migliori se è una macchina a esercitare la classificazione; sia che si tratti di immagini o di video. Tuttavia, si evince come l' accuratezza dei buoni risultati sperimentali da parte di un non umano sono fortemente dipendenti dalla scelta del dataset di immagini a dispozione e da quanto esso sia fornito. Questo elaborato ha lo scopo di riproporre tali studi (pur avendo un dataset relativamente scarno) su un contesto pratico quale avviare un' indagine per la ricerca del probabile *"parente misterioso"* in i *Soliti Ignoti* e riconoscerne il tipo di parentela.

1.1. Due classi di problemi

Il nostro interesse, come già accennato, si pone su due quesiti. *Siamo in grado di determinare, con un alto grado di affidabilità, che due volti dati in input facciano riferimento a individui che sono in parentela?* L' ostacolo principale deriva dal fatto che le possibili relazioni attribuibili fanno parte di un dominio abbastanza ampio; due persone potrebbero essere uno il genitore dell'altro (come in figura 1), fratelli o cugini. Al di là del tipo di parentela, un fattore che aumenta la probabilità di un possibile legame è la similarità dei tratti somatici; molti dettagli sono scrutabili dall' occhio umano, ma con efficacia inferiore rispetto ai calcolatori. Cambiando le specifiche, possiamo spostarci in un' altra classe più semplice: *può un calcolatore riconoscere il tipo di parentela?* Dando per certa la relazione, la possibilità di rientrare in una categoria rispetto ad un' altra è veicolata dal riconoscimento del sesso dei soggetti; il nostro studio canalizza questo problema in 4 classi: *padre - figlia, padre - figlio, madre - figlia, madre -figlio*, relativamente più agevoli da processare rispetto a un rapporto di fratellanza. Il recognizer che implementeremo sarà costretto a prendere una decisione che dichiari una e una sola categoria di appartenenza.

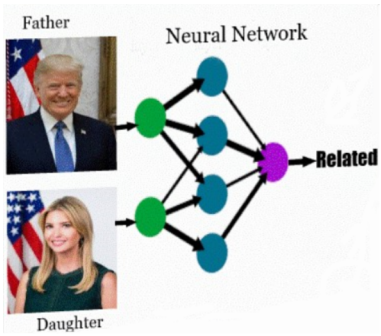

Figure 1. Esempio di parentela padre-figlia.

1.2. Rete Neurale

Il modello computazionale che abbiamo scelto per la risoluzione del compito sono le **reti neurali** (Introduction to artificial neural networks , 2007). Ispiratosi in grandi linee ad una *rete neurale biologica*, questi modelli sono frequentemente utilizzati per risolvere problemi di qualsiasi genere. Si differenzia da soluzioni algoritmiche classiche in quanto chi costruisce il modello non deve dichiarare a priori un set di passi ad hoc fortemente dipendenti dai dati di input: è una struttura che è la generalizzazione universale estesa di qualunque altro modello matematico. L'efficacia delle reti neurali deriva da fasi di training da cui apprendere conoscenza dai dati. Generalmente, una rete di tipo *feed-forward*, come vediamo in figura 2, ha un' architettura partizionata in diversi layers consecutivi che a loro volta sono composti da più nodi (neuroni, fig.3); ogni neurone del layer è collegato a ciascun neurone del layer successivo.

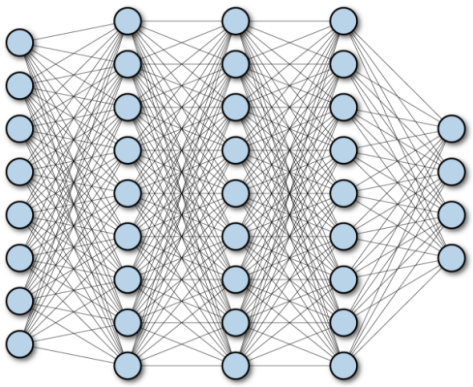

Figure 2. Rete neurale feed forward.

Ogni input viene fatto passare per ciascuno strato; la particolare caratteristica di un neurone è che esso propaga l'informazione in avanti soltanto se supera una specifica *soglia di attivazione*, perciò ciascun nodo ha degli ingressi provenienti dalle elaborazioni di tutti i neuroni dei layers precedenti. Gli archi di collegamento sono pesati e il valore di un nodo è quindi la somma degli ingressi con i pesi più un valore chiamato **bias** (determina se e in quale misura il neurone debba attivarsi o meno).

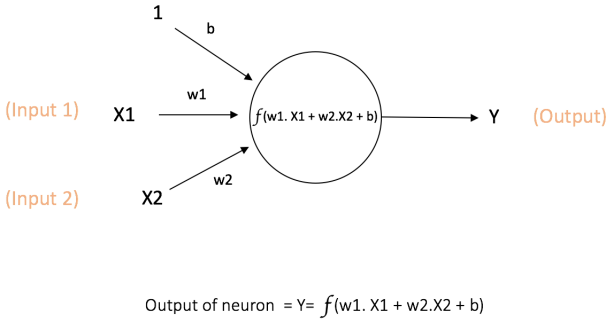

Figure 3. Output di un neurone.

L' intero processo di apprendimento in fig.4 si completa iterando per diverse *epoche* il seguente algoritmo:

(i) *Inizializzazione del modello:* si impostano i pesi in modo casuale a bias a 1.
(ii) *Forward Propagation:* l'input viene fatto passare per tutta la rete per layer successivi.
(iii) *Loss Function:* l'output ricavato dall'ultimo strato si confronta con quello previsto e ne viene calcolato uno scarto (*loss*).
(iv) *BackPropagation:* vengono calcolate le derivate parziali di funzione di costo (che deve essere minimizzata) rispetto a qualsiasi peso e bias della rete.
(v) *Aggiornamento dei pesi*: i pesi vengono modificati e dosati a seconda di un parametro chiamato **learning rate.**

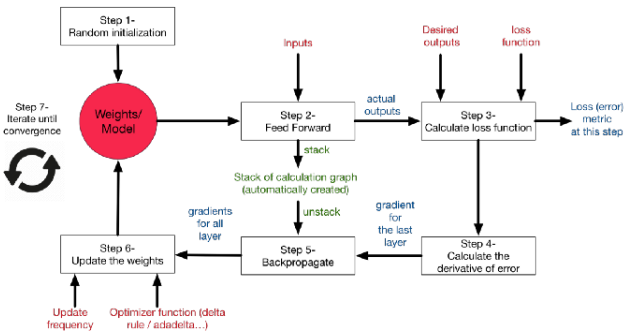

Figure 4. Il processo di apprendimento.

1.3. Rete Neurale Convoluzionale

Nell' ambito della *visione artificiale*, le **reti convoluzionali (CNN)** citate in (Understanding convolutional neural networks with a mathematical model , 2016) sono particolari reti neurali ampiamente utilizzate per il riconoscimento di oggetti in un' immagine. Se dovessimo riconoscere un' entità con una rete *fully connected* (ogni neurone del layer è collegato a ciascun neurone del layer successivo) il compito diventerebbe incredibilmente oneroso. Questa architettura è formata generalmente da diversi livelli (vedi figura 5).

- *Input*: specifica il tensore dell' immagine da dover analizzare.
- *Livello Convoluzionale* ha il compito di individuare pattern dall' immagine tramite un filtro che funge da *sliding windows* che, considerando gruppi di pixel alla volta, rileva angoli, curve, fino a geometrie più complesse.
- *ReLU*: azzera valori non rilevanti ottenuti dai livelli precedenti.
- *Pool*: appartiene alla categoria di filtri che consente la riduzione della dimensione della taglia dell'input cercando di conservare il più possibile le caratteristiche.
- *Fully connected*: è il livello denso di collegamenti tra neuroni che ha lo scopo di identificare la classe di appartenenza dell'input tra quelle proposte con una certa probabilità.

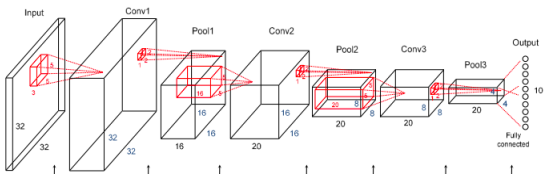

Figure 5. Architettura di una rete convoluzionale.

1.4. Rete Siamese

Una Rete Neurale Siamese (SNN) (David Chicco, Siamese Neural Networks: An Overview , 2020) è formata da una coppia di reti neurali con stessi pesi, parametri e architettura che lavorano su due diversi vettori di input. I vettori di output risultati sono successivamente comparati e ne viene fatto il quadrato della differenza combinando le caratteristiche. Vengono usate in diverse applicazioni, come il riconoscimento di firme (fig.6), check di Questions and Answers simili già presenti in un archivio e, nel nostro caso, per il riconoscimento della parentela. L'output finale non delinea più una probabilità di appartenenza ad una classe, bensì definisce una misura delle distanze da ciascuna delle classi possibili.

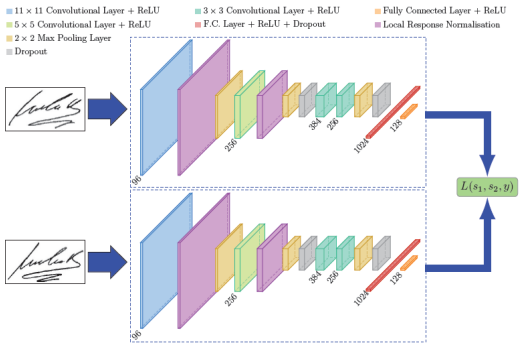

Figure 6. Esempio rete Siamese sul riconoscimento di firme identiche

Una SNN, pur essendo più lenta durante la fase di training, risulta comunque più efficiente rispetto ad una soluzione classica se il dataset è sbilanciato.

1.5. Ambiente di sviluppo

Tutti gli esperimenti del seguente studio sono stati condotti su *Google Colaboratory (Colab)* per avere una semplice integrazione con tutte le librerie Python di stampo Data Science. Di fondamentale importanza è il modulo esterno *PyTorch*, utile per l'elaborazione su tensori usando l'accelerazione delle GPU delle schede grafiche; è stato lo strumento più utilizzato per la costruzione delle nostre reti neurali.

1.6. Dataset

La nostra analisi si sviluppa sul dataset (Recognizing Faces in the Wild , 2019) presente su *Kaggle* per il problema del riconoscimento di persone in parentela. Le immagini dei volti sono raggruppate in folders che rappresentano ciascuna famiglia; ogni cartella contiene ulteriori cartelle con foto della madre, del padre, del figlio e della figlia. Su un csv venogono etichettate tutte le coppie di immagini che sono in relazione e servirà per impostare la fase di validation. Per quanto concerne il problema dell' identificazione del tipo di parentela, si è scelto il dataset (KinFaceW-II , 2015), composto da una raccolta di immagini prese su internet, compresi volti di personaggi pubblici. Nel set di dati KinFaceW-II, ci sono 250 coppie di immagini per ognuna delle quattro relazioni citate in precedenza. Infine, ricordiamo che i volti dei concorrenti

dei *Soliti Ignoti* sono stati estratti da www.raiplay.it/programmi/solitiignoti-ilritorno.

2. UN PRIMO MODELLO

2.1. Una rete "Simple"

In questo capitolo riportiamo i primi passi verso la risoluzione dei due problemi sopra citati. Introduciamo una prima proposta di una rete neurale che chiameremo *SimpleNet* e, come si evince dal nome, l' architettura non è particolarmente complessa. Per quanto riguarda l'individuazione della parentela abbiamo già mostrato come si presta particolarmente bene una rete Siamese; per tal motivo, ciò che faremo è individuare una struttura di base che replicheremo per due **reti CNN**. Quest'ultime dovranno restituire dal proprio layer di output dei tensori, che dovranno concatenarsi in una sequenza e mandati in input ad un susseguirsi di layers pienamente connessi. Di buona norma, dopo ogni strato denso aggiungeremo un livello *ReLU*, ad eccezione dell'ultimo. Fissiamo dei primi parametri, i quali sono:

- **IMG-SIZE** è la dimensione dell'immagine , la quale è stata impostata a 64, perciò in input vengono passate immagini con taglia $64 * 64$.

- **BATCH-SIZE** definisce il numero di campioni alla volta che verranno propagati attraverso la rete per epoca. In genere, usare un BatchSize minore del numero di campioni richiede meno memoria, poiché la procedura di training è meno onerosa. D'altro canto, un BatchSize troppo piccolo diminuirà la stima del gradiente; un mini-batch tende a non far stabilizzare la rete e a portare a risultati non soddisfacenti, come viene mostrato in fig.7. Ci è sembrato opportuno settarlo a 64.

- **NUMBER-EPOCHS** è il numero di epoche che occorrono per il training. 100 epoche per il nostro problema potrebbero essere sufficienti.

- **LEARNING-RATE** è un parametro che precisa con quanta intensità aggiornare i pesi della nostra rete rispetto alla loss. Un tasso di apprendimento troppo grande può far convergere il modello troppo rapidamente verso una soluzione non ottimale, mentre un valore troppo piccolo può causare il blocco del processo di apprendimento (fig.8). È stato fissato a 0.001.

Diamo uno sguardo ora alla composizione delle singole reti. Sicuramente, necessitano di uno strato convoluzionale con il quale dedurre le caratteristiche geometriche dell' immagine; con la funzione *Conv2d*, applichiamo una convoluzione 2D su un segnale di ingresso e fissiamo la dimensione del *kernel di convoluzione* a 3. Inoltre, facciamo confluire lo strato in uno di **ReLU** e applichiamo la **normalizzazione batch**, una tecnica che può migliorare il tasso di apprendimento di una rete neurale. Se abbiamo pesi con incidenza maggiore, anche gli aggiornamenti associati alla retropropagazione sarebbero grandi e viceversa. A causa di questa distribuzione irregolare dei pesi per gli input, l'algoritmo di apprendimento continua a oscillare nella regione del plateau prima di trovare i minimi globali. Per evitare che l'algoritmo di apprendimento passi molto tempo a oscillare nel plateau, normalizzando gli ingressi siamo in grado di portare tutte le caratteristiche degli ingressi alla stessa scala. Aggiungiamo alla fine della catena un layer di **Dropout** (fig.9); esso non propaga attivazioni da un livello precedente con una certa probabilità, rendendo i nodi della rete generalmente più "robusti". La catena formatasi costituisce un primo blocco. Ciò che faremo è ripetere la struttura per ben 3 volte, riducendo man mano la taglia dell' input propagato, per ciascuna sotto rete **CNN**; l'architettura della *SimpleNet* è quella riportata dalla tabella 1.

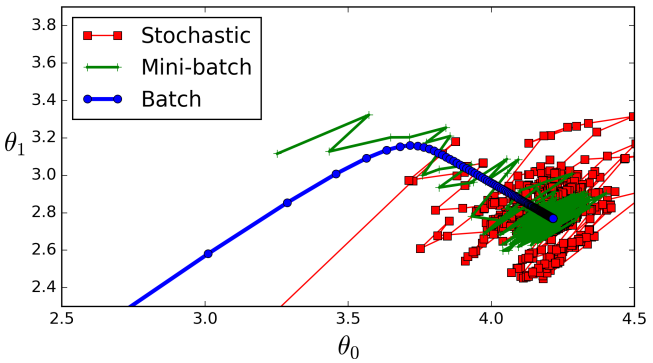

Figure 7. Diverse dimensioni di Batch Size a confronto.

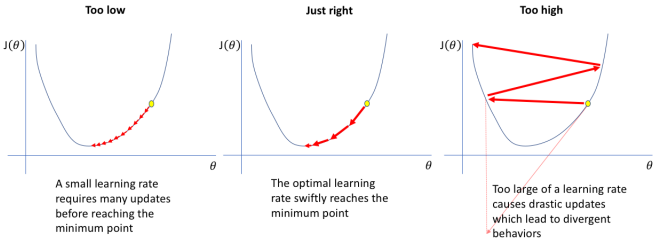

Figure 8. Variazione di Learning Rate.

Table 1. Architettura della SimpleNet.

Layer
(0): ReflectionPad2d((1, 1, 1, 1))
(1): Conv2d(3, 64, kerneSize=(3, 3), stride=(1, 1))
(2): ReLU(inplace=True)
(3): BatchNorm2d(64, eps=1e−05, momentum=0.1, affine=True, track−running−stats=True)
(4): Dropout2d(p=0.2, inplace=False)
(5): ReflectionPad2d((1, 1, 1, 1))
(6): Conv2d(64, 64, kernelSize=(3, 3), stride=(1, 1))
(7): ReLU(inplace=True)
(8): BatchNorm2d(64, eps=1e−05, momentum=0.1, affine=True, track−running−stats=True)
(9): Dropout2d(p=0.2, inplace=False)
(10): ReflectionPad2d((1, 1, 1, 1))
(11): Conv2d(64, 32, kernelSize=(3, 3), stride=(1, 1))
(12): ReLU(inplace=True)
(13): BatchNorm2d(32, eps=1e−05, momentum=0.1, affine=True, track−running−stats=True)
(14): Dropout2d(p=0.2, inplace=False)
(fc1): Linear(inFeatures=640000, outFeatures=500, bias=True)
(fc2): Linear(inFeatures=500, outFeatures=500, bias=True)
(fc3): Linear(inFeatures=500, outFeatures=2, bias=True)

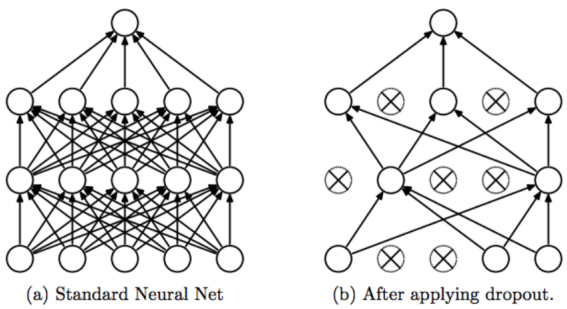

Figure 9. Esempio di Dropout.

2.2. Primi risultati

Dopo aver definito l'intera architettura, siamo passati dalla formalizzazione all' implementazione in Python con l'aiuto di *PyTorch*. Riportiamo in Figura 10 l'andamento della *loss* all' aumentare dei vari BATCH dati in input nel corso delle epoche; il grafico presenta un andamento instabile, senza raggiungere mai un vero e proprio plateau che non converge mai nemmeno facendo proseguire l'esecuzione per il doppio delle epoche. Inoltre, alterando parametri come *Learning Rate* e *Batch Size* non notiamo notevoli cambiamenti. Tuttavia, possiamo affermare che, in fase di validation, l'accuracy oscilla tra il 63% ed il 64%, completando l' addestramento in un' ora.

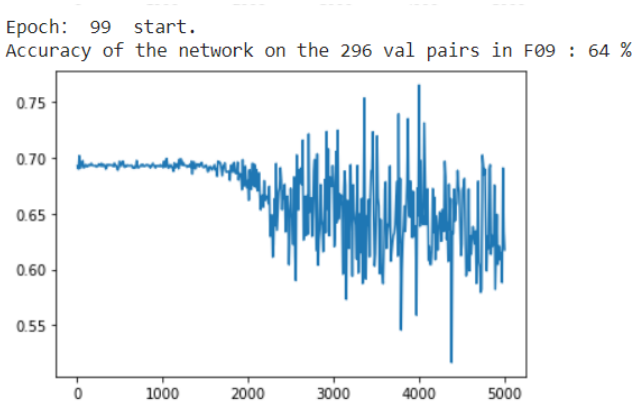

Figure 10. Andamento Loss per SimpleNet.

Per avere una misura affidabile della bontà della rete c'è bisogno di attendere i risultati del testing. A tal proposito, *Kaggle* non fornisce direttamente un csv che denoti delle coppie in parentela o meno, perciò *SimpleNet* ha elaborato le sue predizioni su un set di 200 coppie che sono state caricate successivamente sulla piattaforma, la quale ci ha comunicato l' accuratezza effettiva.

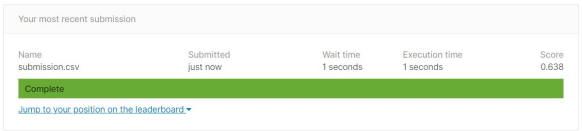

Figure 11. Accuracy calcolata da Kaggle sulla SimpleNet.

Riporta un valore di 63.8% (fig.11), in linea con la validation. Senza modificare in modo radicale la struttura, è il valore più alto che siamo riusciti ad ottenere.

2.3. Transfer learning sul problema a 4 classi

Il secondo obiettivo dell' elaborato è riconoscere il tipo di parentela, il quale differisce in 4 classi. Dando per scontato che una coppia è in relazione, possiamo riadattare l' architettura precedente per risolvere un problema che da binario passa a individuare Madre-Figlio, Madre-Figlia, Padre-Figlio, Padre-Figlia. Questo è possibile grazie al *transfer learning*.

2.3.0.1. *Il* ***Transfer Learning*** *(vedi figura 12)* è una tecnica che permette di riutilizzare i pesi di una rete neurale già addestrata su un problema per risolvere un problema simile. Ciò che si fa è sostituire gli ultimi layer densi, dedicati alla classificazione delle features e rimpiazzarli con dei nuovi layer specifici per il nuovo problema. Questo riduce notevolmente il numero di parametri da utilizzare.

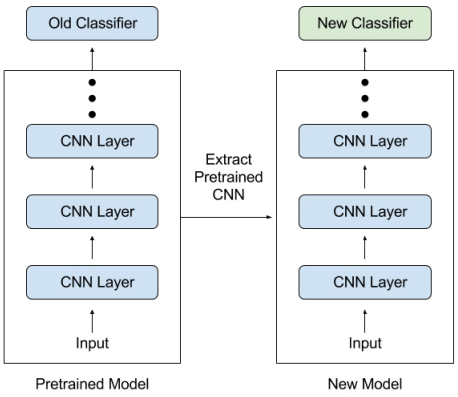

Figure 12. Transfer Learning.

2.3.0.2. ***Adattare il dataset:*** oltre alla sostituzione degli ultimi due livelli e al ridirottamento dell'output da 2 a 4, bisogna ridefinire la logica di "labeling" degli individui dal dataset. *KinFaceW-II* non presenta la classica struttura con csv avente le etichette della classe, bensì dei file .MAT, contenenti array salvati da *MATLAB*. Ciascun file binario contiene delle relazioni con indicazioni per una delle classi; ad esempio, il file "fd-pairs.mat", è suddiviso logicamente in 5 sezioni di taglia simile (quantili), di cui la prima metà di ognuna ha relazioni a True per il ricoscimento *Father-Daughter* e le restanti tuple a False. Con una nuova funzione abbiamo riscritto il codice della classe TrainingDataset in modo che ad ogni epoca venga restituito la tripla (immagine1, immagine3, TipoRelazione).

2.3.0.3. ***Risultati:*** ci si aspetta che l'accuracy sul secondo problema aumenti dopo il transfer learning; d'altronde, classificare degli input su un maggior numero di classi dovrebbe essere un compito meno arduo considerando che si dia per scontato la presenza della parentela. Purtroppo, eseguendo la rete, dopo circa 44 epoche tende ad un plateau al 61% e l'accuracy restituita è del 57%. Un risultato tutt'altro che ottimale. Probabilmente la falla risiede nelle poche immagini fornite dal dataset KinFaceW-II. La scarsa efficacia della SimpleNet ci ha portati alla ricerca di un modello più complesso su cui operare.

2.4. Aumento della complessità della rete

Abbiamo già affermato come assegnando un setting diverso di *BATCH-SIZE, LEARNING-RATE,* o *numero epoche* è pressoché inutile. Per tal motivo, abbiamo effettuato diversi tentativi aggiungendo sia strati pienamente connessi e sia nuovi blocchi convoluzionali. Aumentando il numero di nodi e la taglia del flusso di informazioni che attraversano la rete, non si fa altro che aumentare il costo computazionale, ma l' accuratezza varia e oscilla sempre intorno alla stessa percentuale. Sorprendentemente, una svolta si è avuta grazie a due accorgimenti.

- Non necessariamente più complessità porta a risultati migliori. Per tal motivo, abbiamo ristrutturato i blocchi convoluzionali riducendone il numero di nodi e, allo stesso tempo, abbiamo ridotto il gap di taglia che c'era tra un blocco e l'altro. In questo modo, il passaggio tra un blocco e l'altro risulta più fluido e graduale in funzione della loro dimensionalità.
- Abbiamo eliminato lo strato di *Dropout*. Dovrebbe servire per evitare un possibile overfitting ma, osservando la nostra loss, non dovremmo rientrare in questa casistica. Perciò, eliminare una percentuale di attivazioni di nodi non avrebbe avuto senso. Abbiamo gradualmente ridotto il Dropout finché non abbiamo notato un aumento costante di prestazioni, quindi si è deciso di eliminarlo completamente per tutti i blocchi.

Seguendo questi due principi prettamente empirici, riportiamo nella Tabella 2 l'architettura SimpleNet modificata. Già dall'andamento dell' accuracy in validation vediamo in Figura 14 come raggiunge picchi più alti, indizio che ci ha fatto proseguire su questa linea.

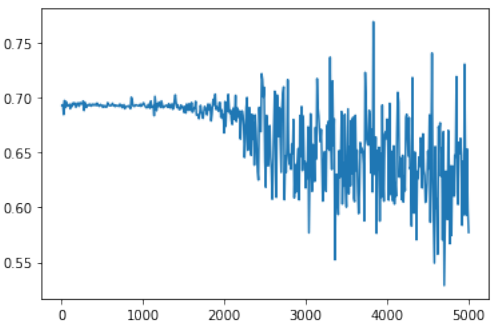

Figure 13. Andamento della loss nella SimpleNet modificata.

A parità di parametri, caricandola su Kaggle restituisce a parità di tempi un' accuracy del 66.8% (fig.15) sul primo problema; un leggero miglioramento rispetto al modello precedente (fig. 13). Se nel primo caso i due accorgimenti hanno avuto un riscontro positivo, sul secondo problema, invece, sia applicando un trasnfer Learning e sia provando

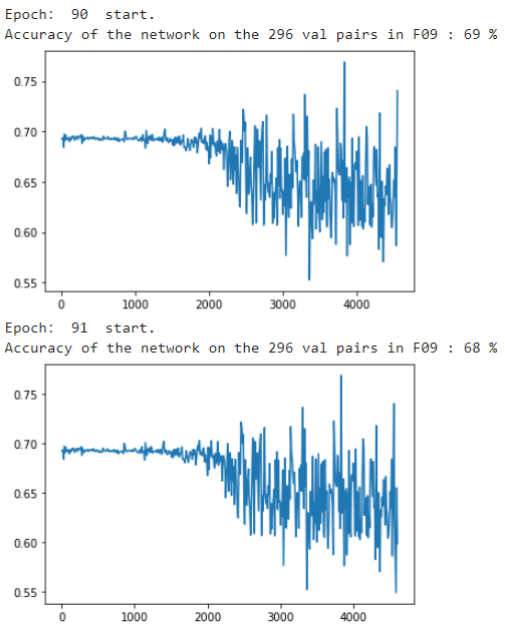

Figure 14. SimpleNet modificata nella fase di training.

la SimpleNet modificata direttamente senza un modello pre-allenato, il testing ci fornisce un 59% di accuracy, soltanto due punti percentuale in più rispetto la versione originaria (Vedi il grafico della loss in figura16).

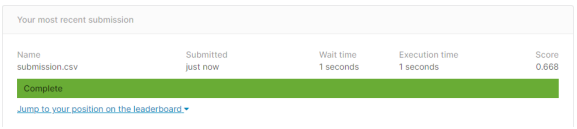

Figure 15. Accuracy della SimpleNet modificata riportata da kaggle.

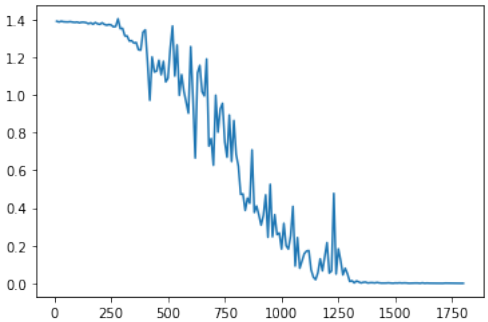

Figure 16. Loss della SimpleNet modificata sul secondo problema.

Table 2. Architettura della SimpleNet modificata.

Layer
(0): ReflectionPad2d((1, 1, 1, 1))
(1): Conv2d(3, 64, kernelSize=(3, 3), stride=(1, 1))
(2): ReLU(inplace=True)
(3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track−running−stats=True)
(4): ReflectionPad2d((1, 1, 1, 1))
(5): Conv2d(64, 32, kernelSize=(3, 3), stride=(1, 1))
(6): ReLU(inplace=True)
(7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track−running−stats=True)
(8): ReflectionPad2d((1, 1, 1, 1))
(9): Conv2d(32, 8, kernelSize=(3, 3), stride=(1, 1))
(10): ReLU(inplace=True)
(11): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track−running−stats=True)
(fc1): Linear(inFeatures=160000, outFeatures=500, bias=True)
(fc2): Linear(inFeatures=500, outFeatures=500, bias=True)
(fc3): Linear(inFeatures=500, outFeatures=2, bias=True)

3. MODELLO DEFINITIVO

3.1. Motivazioni

Abbiamo visto che, dopo alcune modifiche, siamo riusciti ad ottenere un' accuratezza massima del 66% dalla SimpleNet. Abbiamo perciò valutato fosse necessario prendere una nuova rete con caratteristiche completamente diverse e con una struttura tale che ci permettesse di estrarre meglio le informazioni ed aumentare maggiormente l' accuratezza. E' stata individuata una rete di nome Densenet-161, che ci sembrava avere delle ottime caratteristiche per il nostro specifico problema. Essa ha una struttura piuttosto complessa, anche se nella famiglia Densenet sono presenti reti ancora più complesse di essa, che non sono state prese in considerazione per evitare di andare ad utilizzare una rete di dimensioni sproporzionate rispetto al problema da risolvere.

3.2. Architettura Densenet-161

Nella Figura 17 viene mostrata l'architettura Densenet-161 (Kyoung Jun Noh, Jiho Choi, Jin Seong Hong and Kang Ryoung Park, Finger-vein Recognition Based on Densely Connected Convolutional Network Using Score-Level Fusion with Shape and Texture Images , 2017). Il primo blocco arancione è il layer di input, a cui viene fornito un' immagine a tre canali. Inizialmente viene utilizzato un filtro convoluzionale 7x7 per estrarre le informazioni principali, per poi regolare la mappa delle caratteristiche tramite il max pooling. Ciascun blocco di colore blu è un dense block, in cui vengono utilizzati filtri convoluzionali di dimensione 1x1 e 3x3. Sulla base di tale struttura, la dimensione che diventa enorme rispetto alla skip connection della rete viene regolata. Vi è poi

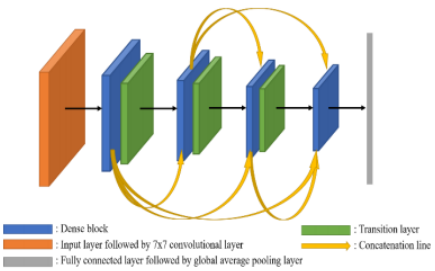

Figure 17. Diagramma dei blocchi di una Densenet-161.

un blocco verde, il livello di transizione, disposto tra i blocchi dense per regolare la dimensione della mappa, al fine di accumulare in modo efficiente le dimensioni. Le frecce gialle, invece, rappresentano le skip connection. La mappa delle caratteristiche di un layer precedente viene concatenata a quella corrente, facendo in modo che le caratteristiche estratte dal livello superiore siano portate in modo efficiente allo strato inferiore. Dopo l'ultimo blocco dense, la classificazione viene completata da uno strato global average pooling di dimensione 8x8, uno strato fully connected ed uno softmax. Nel nostro specifico caso, abbiamo modificato la parte finale della rete, eliminando lo strato fully connected ed inserendone uno nuovo con l'output da noi cercato.

Layer	Filter (number / size)	Input size	Output size
Input layer		224×224×3	224×224×3
conv layer (stride 2)	96 / 7×7	224×224×3	112×112×96
max pool (stride 2)	96 / 2×2	112×112×96	57×57×96
1st Dense block	$6 / \begin{pmatrix} 1\times1 & 4k \\ 3\times3 & k \end{pmatrix}$	57×57×96	57×57×384
Transition layer	$1 / \begin{pmatrix} 1\times1 & 192 \\ 2\times2 & 192 \end{pmatrix}$	57×57×384	29×29×192
2nd Dense block	$12 / \begin{pmatrix} 1\times1 & 4k \\ 3\times3 & k \end{pmatrix}$	29×29×192	29×29×768
Transition layer	$1 / \begin{pmatrix} 1\times1 & 384 \\ 2\times2 & 384 \end{pmatrix}$	29×29×768	15×15×384
3rd Dense block	$36 / \begin{pmatrix} 1\times1 & 4k \\ 3\times3 & k \end{pmatrix}$	15×15×384	15×15×2112
Transition layer	$1 / \begin{pmatrix} 1\times1 & 1056 \\ 2\times2 & 1056 \end{pmatrix}$	15×15×2112	8×8×1056
4th Dense block	$24 / \begin{pmatrix} 1\times1 & 4k \\ 3\times3 & k \end{pmatrix}$	8×8×1056	8×8×2208
global average pooling	2208 / 8×8	8×8×2208	1×1×2208
2D FCL with softmax		1×1×2208	2

Figure 18. Architettura di una Densenet-161.

Nella Figura 18 viene mostrato un dettaglio dell'architettura Densenet-161. Utilizziamo un valore di $k = 48$, identificato come un tasso di crescita della rete. Sulla base del tasso di crescita, il numero di canali della mappa delle caratteristiche di input viene modificato in ciascun blocco dense. Infine, il numero di canali della mappa delle caratteristiche di output nel blocco dense viene determinato dal tasso di crescita. Possiamo notare inoltre la presenza di 4 blocchi dense, ciascuno con rispettivamente 6, 12, 36 e 24 blocchi base.

3.3. Densenet-161 su problema a 2 classi

Consideriamo ora l' integrazione di questa nuova architettura all' interno del nostro progetto andando ad attuare il transfer learning. Il primo step è stato quello di valutarla sul problema di classificazione binaria. Abbiamo dunque caricato il modello densenet-161 in modalità pretrained, ovvero già addestrato sul set di immagini Imagenet, il quale consiste in 1.2 milioni di immagini di training e 50.000 di validation. Come già accennato in precedenza, è stato necessario eliminare lo strato fully connected già presente nella densenet-161, inserendo tre nuovi strati fully connected con output finale pari a 2 classi. Inoltre, abbiamo scelto di utilizzare questa rete come un' estrattore di caratteristiche, andando quindi a congelare i pesi degli strati convoluzionali. Si è scelto anche in questo caso di utilizzare 100 epoche, un batch size pari a 64 ed un learning rate pari a 0.001. I risultati, rispetto alla SimpleNet inizialmente utilizzata sono stati migliori. In fase di validation notiamo un grafico della loss con un andamento decisamente migliore. In fase di testing otteniamo un' accuratezza del 71% in un ora e mezza, andando quindi a migliorare l' accuratezza ottenuta con la SimpleNet, la quale si attestava intorno al 66%.

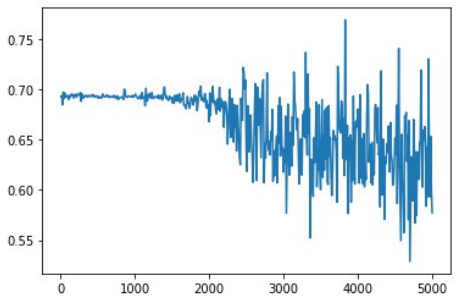

Figure 19. Loss in validation con SimpleNet potenziata

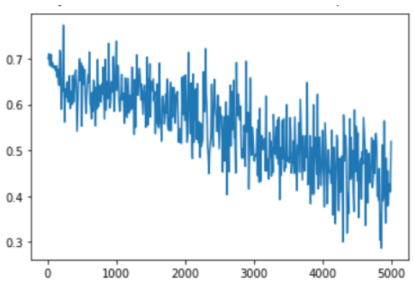

Figure 20. Loss in validation con Densenet-161

In Figura 19 e Figura 20 viene mostrato l'andamento delle funzioni di loss, rispettivamente per la SimpleNet e per la Densenet-161. Possiamo osservare che nella SimpleNet l'andamento è molto irregolare e alla 100-sima epoca la loss non raggiunge valori ottimali, differentemente dalla Densenet-161.

3.4. Densenet-161 su problema a 4 classi

Avendo ottenuto buoni risultati sul problema di classificazione binaria, procediamo quindi con la valutazione della Densenet-161 sul problema di classificazione a quattro classi. Anche in questo caso iniziamo col caricamento della densenet in modalità pretrained, utilizzandola come estrattore di caratteristiche. Ora però ci troviamo ad affrontare un problema differente non solo nel numero di classi di output ma anche nella natura del dataset di training. Ricordiamo, infatti, che nel problema di classificazione binaria possedevamo circa 3300 immagini divise tra training, validation e testing. In questo nuovo problema andiamo a lavorare con 1000 immagini comprendendo tutti e quattro i tipi di relazione. Possiamo assumere che si tratta di un problema di difficoltà inferiore rispetto al problema di riconoscimento binario, ma in contrapposizione abbiamo il problema di avere a disposizione un dataset di dimensioni ridotte. Eseguiamo perciò alcune modifiche preliminari, andando a ridurre la misura del batch size da 64 a 25 e cercando di trovare un numero ottimale di epoche di addestramento, modificando inizialmente il suo valore da 100 a 60. Lasciamo infine inalterato il valore del learning rate impostato a 0.001.

3.4.1. Primo test

Sulla base dei parametri settati, verifichiamo il comportamento della rete in fase di validation e testing. Le 1000 immagini a disposizione, sono organizzate in 5 quantili da 200 immagini ciascuno, secondo la natura del dataset che abbiamo preso in considerazione. Strutturiamo perciò la ripartizione dei quantili in questo modo:

- **Training**: quantili 1,2,3
- **Validation**: quantile 4
- **Testing**: quantile 5

Prima di raggiungere un plateau ci fermiamo all' epoca numero 39 ed otteniamo (fig.21 in basso):

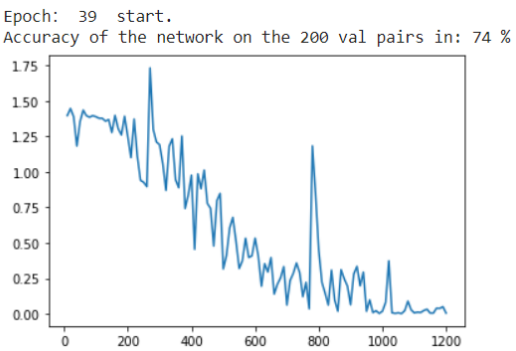

Figure 21. Loss in validation su quantile numero 4

Osserviamo un buon comportamento in fase di validation. Verifichiamo ora cosa accade in fase di testing. Di seguito viene mostrata la matrice di confusione ottenuta dopo il testing:

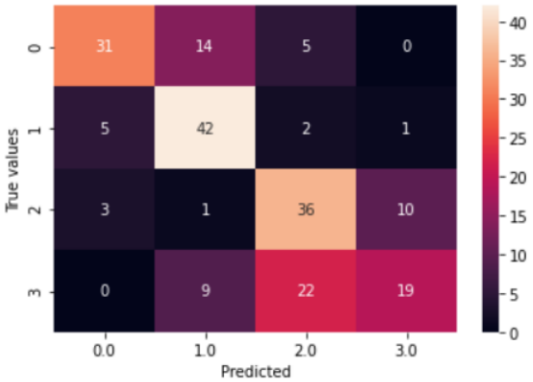

Figure 22. Matrice di confusione su testing quantile numero 5

Assumiamo che la classe father-daughter corrisponda al valore 0, la classe father-son al valore 1, la classe mother-daughter al valore 2 ed infine quella mother-son al valore 3. Possiamo notare, nella matrice di confusione in Figura 22 una particolare difficoltà della rete nel riconoscere la parentela di tipo mother-son. Ne deriva in definitiva un' accuratezza totale del 64%. Decidiamo perciò a questo punto di eseguire un ulteriore test invertendo però i quantili di validation e testing. In sostanza la nostra rete verrà addestrata sugli stessi quantili precedenti ma verrà testata su quantili invertiti.

3.4.2. Secondo test

Ripartiamo i quantili nel seguente modo:

- **Training**: quantili 1,2,3
- **Validation**: quantile 5
- **Testing**: quantile 4

Riportiamo i risultati in fase di validation prima della raggiunta di un plateau (fig. 23). Fermandoci all'epoca numero 30 otteniamo:

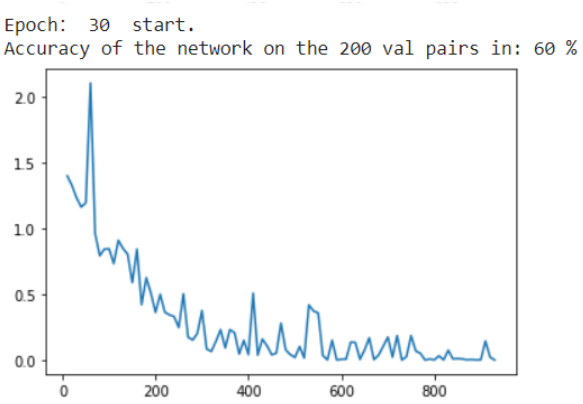

Figure 23. Loss in validation su quantile numero 5

Notiamo un' accuracy minore in fase di validation rispetto al primo test effettuato. Vediamo ora il comportamento della matrice di confusione e l' accuratezza del testing.

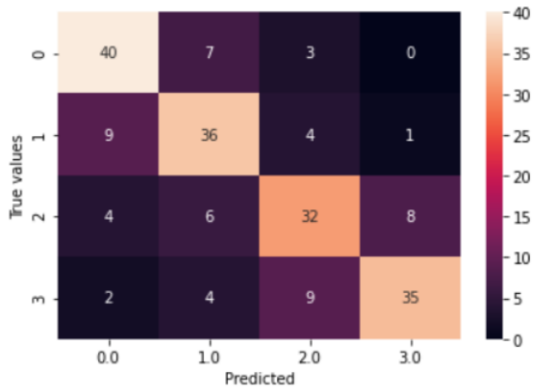

Figure 24. Matrice di confusione su testing quantile numero 4

La matrice di confusione(Figura 24) ha una ripartizione più omogenea dei valori ed in questo caso la rete non va in confusione sulla parentela di tipo mother-son. La rete raggiunge un' accuratezza del 71,5% in fase di testing, sebbene in fase di validation i risultati erano pessimi.

3.4.3. Considerazioni

Analizziamo ora i risultati precedentemente ottenuti, prima di procedere ad ulteriori test. Abbiamo constatato che allenando la rete sulle stesse immagini di training, essa presenta risultati di testing ben differenti scambiando i quantili riservati alla fase di validation ed alla fase di testing. Presumiamo che questo sia dovuto alle dimensioni ridotte del dataset che stiamo utilizzando e alla diversa difficoltà che può essere presente in alcune immagini rispetto ad altre. Crediamo, perciò, sia limitante assegnare un valore di accuratezza ben preciso ad una rete neurale sulla base di queste motivazioni. A rigor di prova abbiamo rieseguito diverse volte le due tipologie di esperimenti discusse precedentemente. Riportiamo i risultati:

Training	Accuratezza test1	Accuratezza test2
1	64%	71,5%
2	65%	69%
3	60,5%	77%
4	62%	79%
5	63%	74%

Table 3. Due tipologie di testing a confronto

In definitiva, in tabella 3 possiamo osservare una forte variabilità della rete, anche sulla stessa rete allenata più volte.

4. CROSS VALIDATION

L instabilità della rete può essere colmata ampliando il numero di immagini passate nel training. Non avendo fonti esterne da far confluire nel dataset, abbiamo svolto diversi esperimenti tentando di inglobare anche il quantile di validation nella fase di allenamento. In questo modo riusciamo ad aumentare il volume di dati su cui lavorare del 20% . Prendendo in esame la casistica in cui i quantili di training siano 1-2-3-4 e 5 per il testing, si osserva come l' accuratezza è di circa del 64%. Anche qui, vi è un discorso analogo su come permutando i quantili si possa avere risultati altamente variabili.

4.0.0.1. ***Cross Validation*** è una tecnica statistica che permette una validazione "dinamica" sul set di dati,(validazione statica in Figura 25), ossia, usare in modo alternato ciascun quantile per il testing cambiandolo con uno di training coprendo tutte le possibili permutazioni. Si parla infatti di *k-fold cross validation*(Figura 26) indicando la divisione del dataset in k parti uguali.

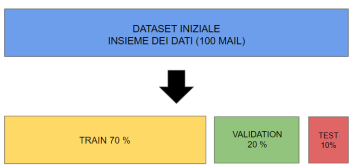

Figure 25. Validazione statica.

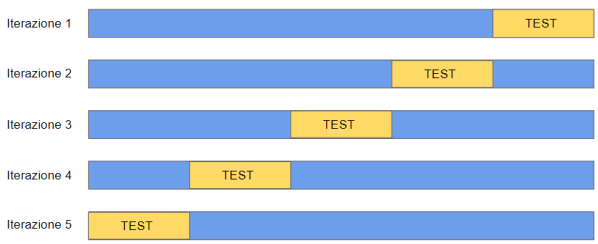

Figure 26. K-fold validation.

Seguendo questo filo conduttore, abbiamo sperimentato tutti i possibili casi, sempre sacrificando il quantile di validation.

4.0.0.2. ***Training 1-2-3-4, testing: 5*** Nella disposizione canonica dei quantili, abbiamo un accuracy del 64%. Maggiori difficoltà sono state riscontrate nel riconoscere le parentele madre-figlia, confuse con la parentela madre-figlio.(Figura 27)

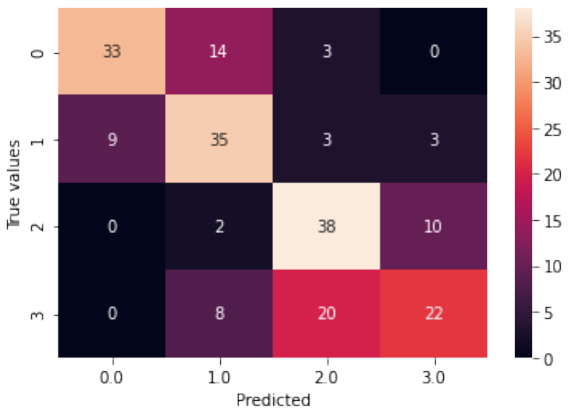

Figure 27. Training 1-2-3-4, testing: 5, accuracy: 64%

4.0.0.3. ***Training 1-2-3-5, testing: 4*** È la permutazione che ha ottenuto performance migliori con un accuracy del 77%. La distribuzione delle predizioni è nettamente più omogenea rispetto al caso precedente.(Figura 28)

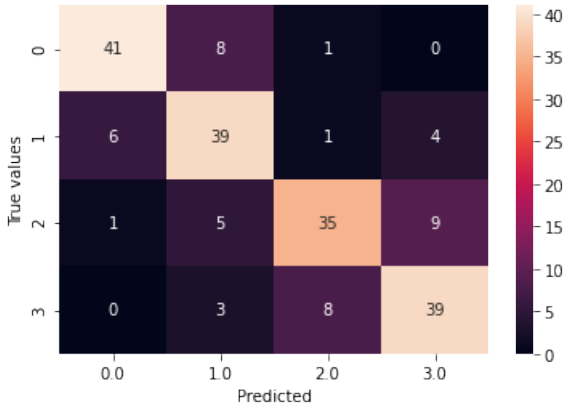

Figure 28. Training 1-2-3-5, testing: 4, accuracy: 77%

4.0.0.4. ***Training 1-2-4-5, testing: 3*** Anche qui, le relazioni madre-figlia e madre-figlio sono maggiormente confuse. (Figura 29)

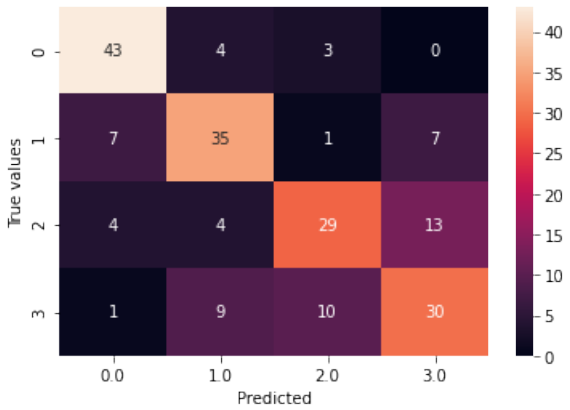

Figure 29. Training 1-2-4-5, testing: 3, accuracy: 68.5%

4.0.0.5. ***Training 1-3-4-5, testing: 2*** Con un accuratezza del 73%, è la seconda permutazione più prestante. Errori di labeling si verificano in modo più incidente nella parentela padre-figlia con padre-figlio, e la relazione madre-figlia confusa con madre-figlio. (Figura 30)

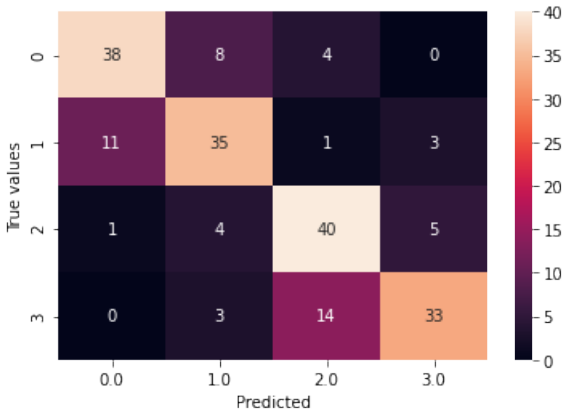

Figure 30. Training 1-3-4-5, testing: 2, accuracy: 73%

4.0.0.6. ***Training 2-3-4-5, testing: 1*** Considerazione analoga alla precedenti. Un forte peso che ha destabilizzato il processo di recognition viene riscontrato nel riconoscere la relazione padre-figlio scambiata per padre-figlia. (Figura 31)

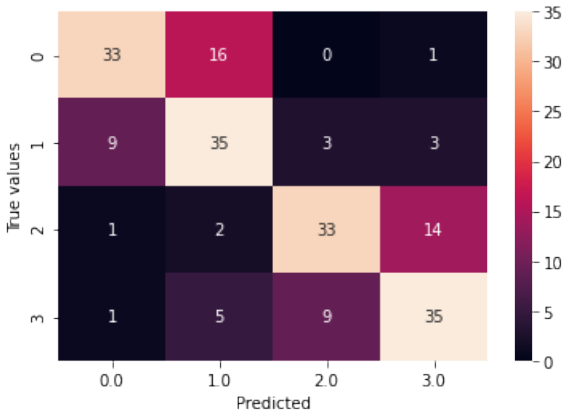

Figure 31. Training 2-3-4-5, testing: 1, accuracy: 68%

4.0.0.7. ***Conclusioni:*** Possiamo affermare che le difficoltà dei modelli hanno sede nel riconoscimento del sesso del figlio. Quasi in tutti i casi, invece, il sesso del genitore è dedotto più facilmente. Tutti e 5 i modelli saranno usati successivamente per il riconoscimento degli ignoti. Riportiamo in tabella 4 i modelli sperimentati fin ora con le relative accuratezze.

Rete	Problema	Test Accuracy	Dataset
SimpleNet	2 classi	63.8%	FIW
SimpleNet	4 classi	57%	Kinface
SimpleNet v.2	2 classi	66.8%	FIW
SimpleNet v.2	4 classi	59%	Kinface
Densenet-161	2 classi	71%	FIW
Densenet-161	4 classi	68-77%*	Kinface

Table 4. Confronto tra modelli.
**Variabile in funzione delle permutazioni e dal numero di quantili usati nel training.*

5. TEST SU SOLITI IGNOTI

In questa sezione andremo a testare la nostra rete su alcune puntate del programma televisivo *I soliti ignoti*. Si tratta di un game show in cui un concorrente, data una certa persona, deve indovinare chi tra 8 possibili persone è imparentata con questa persona fissata a priori, sapendo che solo una tra le 8 persone lo è. Ispirandoci a quella che è la natura di questo gioco procederemo con due test sequenziali, prima utilizzando la rete di riconoscimento della parentela e poi successivamente quella di riconoscimento del tipo di parentela. Sarà prima però necessario estrarre dalle puntate alcuni volti che andranno a comporre il nostro dataset di testing.

5.1. Estrazione volti

Per quanto concerne l' estrazione dei volti, abbiamo selezionato alcune puntate dal sito `www.raiplay.it/programmi/solitiignoti-ilritorno` e catturato i frame di nostro interesse. Ciascun frame è stato poi dato in input ad un programma di estrazioni di volti da noi realizzato, sfruttando le potenzialità delle librerie **openCV** e **dlib**. In un primo momento è stata eseguita un' operazione di detection dei volti presenti all'interno dell'immagine. Avendo poi a disposizione le coordinate top, down, left e right del rettangolo che racchiudeva i volti, è stato poi possibile eseguire il ritaglio e il successivo inserimento nel nostro set di testing.

5.2. Riconoscimento parentela

Come detto in precedenza, il nostro primo test sarà quello rivolto al problema di classificazione binario; ovvero riconoscere date due persone in input se esse sono imparentate o meno. A tal proposito andremo a selezionare due puntate: una in cui il concorrente riesce ad indovinare il parente misterioso ed un' altra in cui non indovina. Analizziamo dunque il comportamento della rete su queste due puntate.

5.2.1. Concorrente non indovina il parente misterioso
Prendiamo in esame la puntata del 04/05/2021 in cui il concorrente non indovina il parente misterioso. Estraiamo tutti gli 8 possibili parenti, di cui uno solo sarà effettivamente quello corretto. A questo punto creiamo 8 coppie formate dal parente misterioso e tutte le possibili persone imparentate con esso. Su questo input otteniamo il seguente risultato:

	img1	img2	Parentela	Predizione
0	4maggio[2]3	4maggio[2]1	1	0.0
1	4maggio[2]3	4maggio11	0	1.0
2	4maggio[2]3	4maggio21	0	1.0
3	4maggio[2]3	4maggio31	0	0.0
4	4maggio[2]3	4maggio41	0	0.0
5	4maggio[2]3	4maggio51	0	1.0
6	4maggio[2]3	4maggio61	0	1.0
7	4maggio[2]3	4maggio71	0	0.0

Figure 32. Risultati della rete quando il concorrente sbaglia

Nella prima riga è stata posta la reale parentela. In Figura 32 possiamo osservare, quindi, come così come per il concorrente, anche la rete non riesce ad indovinare il parente misterioso, notando invece alcune somiglianze con altre persone non imparentate.

5.2.2. Concorrente indovina il parente misterioso
Consideriamo ora la puntata del 17/05/2021 in cui il concorrente indovina il parente misterioso. Mostriamo il comportamento della rete su questa puntata:

	img1	img2	Parentela	Predizione
0	17maggio[3]1	17maggio[3]3	1	1.0
1	17maggio[3]1	17[maggio]11	0	1.0
2	17maggio[3]1	17[maggio]21	0	1.0
3	17maggio[3]1	17[maggio]31	0	0.0
4	17maggio[3]1	17[maggio]41	0	0.0
5	17maggio[3]1	17[maggio]51	0	0.0
6	17maggio[3]1	17[maggio]61	0	0.0
7	17maggio[3]1	17[maggio]71	0	1.0

Figure 33. Risultati della rete quando il concorrente indovina

In questo caso, osserviamo in Figura 33, che così come il concorrente, la rete indovina il parente misterioso, con un ridotto numero di falsi positivi.

Figure 34. Coppia padre-figlia

Notiamo che si tratta di una relazione di tipo padre-figlia(Figura 34), per la quale verificheremo poi in seguito la corrispondenza tramite la rete che classifica sui tipi di parentela.

5.3. Riconoscimento tipo di parentela

Per eseguire questa tipologia di test consideriamo 20 puntate. Per ogni puntata andremo a prelevare la coppia di persone che corrispondono alla reale parentela. Analizzeremo poi il comportamento della nostra rete ideata per la classificazione sul tipo di parentela. Ricordiamo ora le 4 possibili classi: father-daughter(fd), father-son(fs), mother-daughter(md), mother-son(ms). Nella tabella in Figura 35 vengono mostrati i risultati ottenuti. Avendo attuato la cross validation abbiamo a disposizione 5 modelli diversi da poter provare sul nostro set di testing. Le accuratezze, mostrate in basso nella tabella, ottenute dai diversi modelli sulle 20 puntate testate variano dall' 80% al 90%. Anche in questo caso possiamo osservare una certa variabilità nei risultati anche se non con differenze marcate. Notiamo, inoltre, che andando ad eseguire il test sulle puntate dei Soliti Ignoti l' accuratezza ha subito un ulteriore incremento. C' è da dire che le immagini di questo nuovo test hanno una qualità migliore di quelle utilizzate nel dataset usato per costruire la rete. Inoltre si tratta di un test su un numero limitato di coppie di immagini, motivo per il cui risulta difficile fare un confronto. Per quanto riguarda la coppia padre-figlia(puntata 17maggio) per la quale era stata precedentemente individuata una parentela, qui viene definita come padre-figlia in tre modelli su cinque.

fd = 0 fs = 1 md = 2 ms = 3

img1	img2	Parentela	2345(68 %)	1234(64 %)	1235(77%)	1345(73 %)	1245 (68.5%)
1Aprile[1]1	1Aprile[1]2	1	1	1	1	1	1
4Aprile[8]3	4Aprile[8]1	0	0	0	0	0	0
9Aprile[8]3	9Aprile[8]1	1	1	1	1	1	1
11Aprile[2]3	11Aprile[2]2	2	2	2	2	2	2
14Aprile[7]2	14Aprile[7]1	0	0	0	0	0	0
16Aprile[3]1	16Aprile[3]2	0	0	0	0	0	0
18Aprile[6]1	18Aprile[6]2	2	2	2	2	2	2
19Aprile[3]3	19Aprile[3]2	2	2	2	2	2	2
20Aprile[2]2	20Aprile[2]3	1	1	1	1	1	0
23Aprile[5]3	23Aprile[5]1	0	1	1	1	1	0
4maggio[2]1	4maggio[2]3	0	1	1	1	1	1
5maggio[3]1	5maggio[3]2	1	1	1	1	1	1
8maggio[1]3	8maggio[1]1	1	3	1	3	3	1
11maggio[3]2	11maggio[3]3	3	3	3	3	3	3
17maggio[3]3	17maggio[3]1	0	1	1	0	0	0
22marzo[3]3	22marzo[3]2	0	0	0	0	0	0
23marzo[4]2	23marzo[4]1	3	3	3	3	3	3
24marzo[1]2	24marzo[1]1	0	0	0	0	0	0
26marzo[2]1	26marzo[2]2	0	0	0	0	0	0
29marzo[3]2	29marzo[3]1	3	3	3	3	3	3
			80%	85%	85%	85%	90%

Figure 35. Testing sul riconoscimento del tipo di parentela

6. CONCLUSIONI E SVILUPPI FUTURI

Concludiamo riassumendo quelli che sono i punti cardine individuati durante lo sviluppo di questo progetto. Abbiamo osservato quanto molto incida la dimensione di un dataset sui risultati ottenuti, in particolare nel nostro caso ci siamo trovati di fronte ad una consistente variazione nei livelli di accuratezza. Altro fattore particolarmente incidente è la difficoltà delle immagini che si vanno a testare. Riteniamo molto soddisfacente, ad ogni modo, il comportamento della rete Densenet-161 sulla risoluzione del problema di riconoscimento del tipo di parentela. Abbiamo notato che la maggior parte dell' errore commesso risiede nella discriminazione del sesso del figlio, individuando correttamente il sesso dei genitori.
Per quanto riguarda gli sviluppi futuri proponiamo l' approfondimento di alcuni aspetti su cui pensiamo sia possibile agire positivamente. Per quanto concerne la prima rete di classificazione binaria abbiamo analizzato il seguente aspetto. Partiamo dal presupposto che si tratta di una rete costruita per andare a risolvere il problema dei "Soliti Ignoti". Basandoci sulla natura del gioco ci domandiamo, cosa accadrebbe se invece di prendere come output dalla rete i valori 0 ed 1 si prendesse la percentuale di quanto la rete crede le due persone siano parenti? Un po' come una persona che nel tentare di indovinare assegna un grado di somiglianza e poi sceglie quella che ritiene più somigliante. Un altro aspetto da poter affrontare è quello di ampliare il tipo di classificazione della rete, ad esempio, alle parentele del tipo fratello-sorella. Un ulteriore sviluppo potrebbe essere quello di rendere la rete discriminante sul fattore età, in modo da trovarsi agevolata poi nel processo di riconoscimento.

RIFERIMENTI

Bordallo Lopez, Miguel and Hadid, Abdenour and Boutellaa, Elhocine and Goncalves, Jorge and Kostakos, Vassilis and Hosio, Simo. Kinship verification from facial images and videos: human versus machine (2018)

Grossi, Enzo; Buscema, Massimo, Introduction to artificial neural networks (2007)

C.-C. JayKuo, Understanding convolutional neural networks with a mathematical model (2016)

David Chicco, Siamese Neural Networks: An Overview (2020)

Northeastern SMILE Lab, Families In the Wild: A Kinship Recognition Benchmark (2019)

Jiwen Lu, Junlin Hu, KinFaceW-II (2015)

Kyoung Jun Noh, Jiho Choi, Jin Seong Hong and Kang Ryoung Park, Finger-vein Recognition Based on Densely Connected Convolutional Network Using Score-Level Fusion with Shape and Texture Images (2017)