Emotion recongnition con ConvLSTM2D

Alessio Ambruoso
Dipartimento di informatica (LM-18)
Sicurezza informatica
Scafati, Italia
a.ambruoso1@studenti.unisa.it

Marco Borrelli
Dipartimento di informatica (LM-18)
Sicurezza informatica
Somma Vesuviana, Italia
m.borrelli18@studenti.unisa.it

RobertoVeneruso
Dipartimento di informatica (LM-18)
Sicurezza informatica
Scafati, Italia
r.veneruso1@studenti.unisa.it

I. INTRODUZIONE

Riconoscere automaticamente le espressioni facciali è diventata un'importante area di ricerca che coinvolge machine learning, computer vision e behavioral sciences.
Per fare ciò, si ha bisogno di creare un dataset, il quale rappresenta una collezione di dati strutturati ed elaborati in modo tale da poter essere processati da un algoritmo. Il riconoscimento delle emozioni può essere utilizzato in diversi ambiti come quello della sicurezza, delle interazioni uomo-macchina e dell'assistenza sanitaria.
In particolare, l'obiettivo del progetto è quello di addestrare una rete neurale al riconoscimento di un dataset di espressioni facciali tramite la Blob Detection.

II. DATASET

Per lo sviluppo dell'applicazione realizzata è stato utilizzato il dataset "The Extended Cohn-Kanade" (CK+). Esso contiene 593 video sequenze, in cui ogni frame ha risoluzione 640x490 o 640x480 pixels. Si hanno in totale 123 soggetti differenti, che vanno dai 18 a 50 anni d'età di genere e provenienza diverse. Di questi video, 327 sono stati catalogati in una delle sette classi di emozioni: sorpresa, tristezza, felicità, paura, disgusto, disprezzo, rabbia. (Fig.1)

Fig 1: Sette classi di emozioni

III. ELABORAZIONE DELLE IMMAGINI

L'elaborazione delle immagini viene utilizzata principalmente per estrarne diverse caratteristiche. Poiché le immagini digitali contengono diversi oggetti e informazioni, possiamo eseguire varie tecniche per individuare e rilevare tali dati. Una di queste è la Blob Detection.
Nella visione artificiale, la Blob Detection è una tecnica che ha come obiettivo di rilevare punti e/o regioni in una immagine che differiscono in proprietà come luminosità o colore se comparate con l'ambiente. Nel lavoro di preparazione il rilevamento blob è utilizzato per ottenere regioni di interesse per ulteriori processi di lavoro.
Lo sviluppo iniziale del progetto sull'elaborazione delle immagini è stato diviso in 4 step (fig.2):

1. FACE DETECTION
2. DIFFERENCES
3. THRESHOLDING
4. ERODE

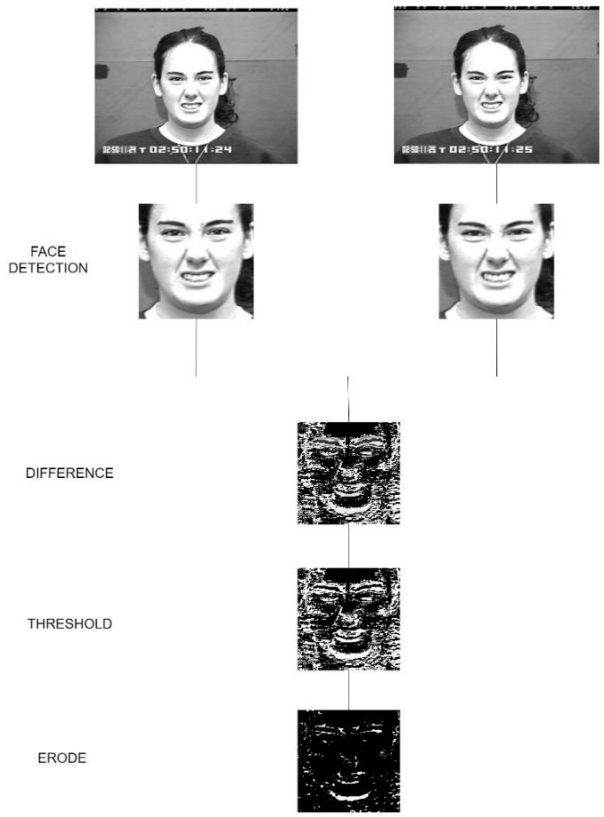

Fig 2: Output delle fasi di elaborazione delle immagini

1. La face detection è una tecnica di intelligenza artificiale, utilizzata in biometria per identificare o verificare l'identità di una persona a partire da una o più immagini che la ritraggono. Nel progetto, la face detection è stata utilizzata per rilevare il volto dei soggetti in ogni frame delle video sequenze, come descritto nello script. (fig.3)

```
for img in imgList:
    img_gray = cv2.imread('cohn-kanade-images/'+x+'/'+y+'/'+img, 0)
    detector = dlib.get_frontal_face_detector()
    faces = detector(img_gray, 1)
    for face in faces:
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()
    crop_img= img_gray[y1:y2, x1:x2]
    crop_img= cv2.resize(crop_img, (267,267))
```

Fig.3: Script face detector

In questo modo estraiamo dai vari frame i soli volti dei soggetti uniformando le dimensioni in 267x267 pixels ed eliminando le informazioni in eccesso dell'immagine.

2. La differenza tra le immagini ha permesso di confrontare frame successivi di una video sequenza in modo tale da avere delle nuove immagini risultanti che mostrassero il movimento del volto per il raggiungimento dell'emozione. La differenza è stata eseguita su ogni coppia successiva di frame di ogni video sequenza in modo tale da riuscire a collezionare più dati possibili sul cambiamento dell'espressione facciale di ogni individuo.

3. La funzione threshold fa parte della libreria software openCV ed è stata utilizzata per trasformare un'immagine in scala di grigi ad un'altra in cui i valori dei pixel siano binari, cioè o con valore 0 o con valore 255. La funzione prende come primo argomento l'immagine in scala di grigi, come secondo il valore di threshold, sotto il quale i pixel assumono il valore 0 e sopra il quale assumono il valore massimo specificato come terzo parametro della funzione. L'ultimo parametro indica il tipo di thresholding da applicare. In questo modo durante l'addestramento della rete, a seguito di una sostituzione dei valori 255 con il valore 1, l'immagine risultante diviene facilmente processabile da una rete neurale avendo solo due valori possibili.

4. La funzione erode anch'essa fa parte della libreria openCV ed è stata utilizzata per ridurre attraverso un kernel le differenze in eccesso tra i vari frame delle video sequenze, in modo da non avere troppa dispersione delle informazioni.

Gli step 2,3,4 sono descritte nello script. (fig.4)

```
2. new_img= tempImage-crop_img
3. _,img_thre= cv2.threshold(new_img,127,255,cv2.THRESH_BINARY)
   kernel = np.ones((5, 5), np.uint8)
4. img_thre=cv2.erode(img_thre, kernel)
```

Fig. 4: Gli script dei 4 step

I risultati finali di questi step sono stati inseriti in un nuovo dataset chiamato per convenzione '*binary*'.

Questo è stato poi elaborato in modo da estrarne solo le video sequenze catalogate dalle emozioni ed è stato suddiviso in base alla varie emozioni. (fig.5)

```
dirList= next(os.walk('binary'))[1]
for x in dirList:
    imgList=next(os.walk('binary/'+x))[2]
    for img in imgList:
        subfolder=img[5:8]
        print(x+"_"+subfolder)
        listFiles= next(os.walk("Emotion/"+x+"/"+subfolder))[2]
        if listFiles:
            f=open("Emotion/"+x+"/"+subfolder+"/"+listFiles[0], "r")
            emotion=f.read()[3:4]
            nameFolder=x+"_"+subfolder
            if not os.path.exists("dataset/"+emotion+"/"+nameFolder):
                os.makedirs("dataset/"+emotion+"/"+nameFolder)
            copyfile("binary/"+x+"/"+img, "dataset/"+emotion+"/"+nameFolder+"/"+img)
```

Fig.5: Script per dataset organizzato per emozioni

Il dataset così prodotto è stato poi elaborato ulteriormente per bilanciare la quantità di video presente per ogni emozione e la quantità media di frame per ogni video. In questo modo si evita che la rete si discosti da un valore reale e che propendi verso l'emozione con il maggior numero di video sequenze. Al termine del bilanciamento il dataset è costituito da 7 emozioni di cui ognuna con 15 video sequenze composte in media da 20 frame.

IV. RETE NEURALE

È stata utilizzata la rete neurale ConvLSTM2D, la quale risulta molto simile ad una LSTM, ma le trasformazioni di input e le trasformazioni ricorrenti sono entrambi convoluzionali, questo vuol dire che la rete utilizza la convoluzione per processare i dati in input ed inserisce il tutto in un modulo LSTM. Inoltre, la rete è bidimensionale perché il kernel scorre lungo due dimensioni sui dati.

I modelli di rete convoluzionale sono stati sviluppati per problemi di classificazione delle immagini dove il modello apprende una rappresentazione interna di un input in un processo denominato apprendimento delle caratteristiche, eliminando la necessità di estrarre manualmente le feature. Questi modelli utilizzano dei kernel convoluzionali grazie ai quali si riescono ad ottenere dei filtri sulle immagini che identificano e mettono in risalto determinate caratteristiche, mentre ne nascondono altre.

LSTM (Long short-term memory) è un tipo di rete neurale ricorrente capace di apprendere e ricordare su lunghe sequenze di dati in input. Una delle caratteristiche principali è l'uso delle informazioni elaborate precedentemente per il calcolo della prossima, un esempio è quello dei frame di un video e di come la rete utilizzi i frame precedentemente processati per predire il prossimo frame.

È stato deciso di utilizzare la rete neurale ConvLSTM2D perché sfrutta le capacità della rete LSTM che è in grado di elaborare efficientemente le serie temporali di immagini come le video sequenze presenti nel nostro dataset, ma allo stesso tempo utilizza le proprietà della rete convoluzionale che riesce ad estrarre dai frame elaborati con la Blob Detection, attraverso vari filtri, le caratteristiche più

importanti di ognuno, in modo da riconoscere correttamente la classe di provenienza. (Fig.6, Fig.7)

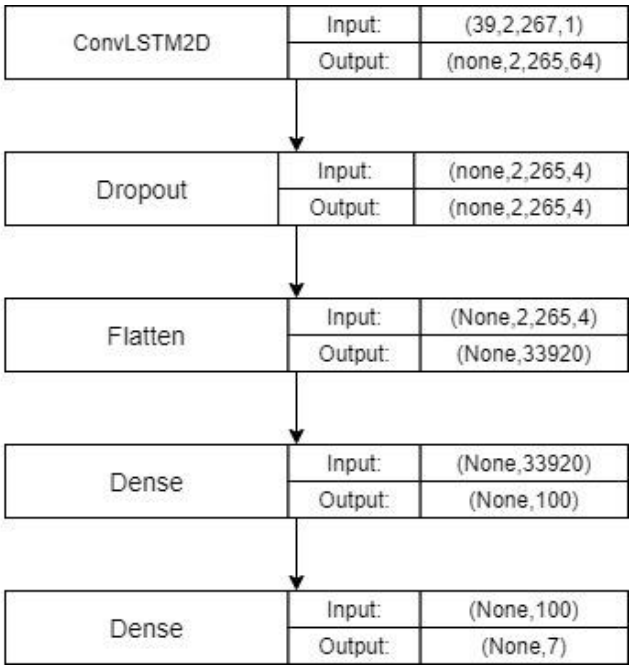

Fig.6: Rete Neurale

```
def evaluate_model(trainX, trainy, testX, testy):
    # define model
    print(trainX.shape)
    verbose, epochs, batch_size = 1, 100, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
    trainy.shape[1]
    # reshape
    n_length = 267
    trainX = trainX.reshape((trainX.shape[0], n_timesteps, n_features, n_length, 1)
    testX = testX.reshape(testX.shape[0], n_timesteps, n_features, n_length, 1)
    # define model
    model = Sequential()
    model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu',
    input_shape=(n_timesteps, n_features, n_length, 1)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
    metrics=['accuracy'])
    # fit network
    history= model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
    verbose=verbose, validation_data=(testX, testy))
    model.summary()

    # evaluate model
    y_pred = model.predict(testX, batch_size)
    y_pred= np.argmax(y_pred, axis=1)
    y_label= np.argmax(testy, axis=1)
    print('Classification Report')
    print(classification_report(y_label, y_pred, target_names=rightLabels))
    print('Confusion Matrix')
    plot_confusion_matrix(confusion_matrix(y_label, y_pred), rightLabels)

    print("finish")
    return history
```

Fig. 7: Script modello

Il primo layer ConvLSTM2D crea uno stato LSTM convoluzionale 2D dove:

- **Filters** è la dimensione dello spazio di output (numero di filtri in output nella convoluzione).
- **Kernel_size** è la dimensione della finestra di convoluzione che di solito viene impostata a 3x3, ma in questo caso, a causa dei frame di dimensione 2x267, è stata impostata a 1x3 per migliorare il processo di convoluzione.
- **Input_shape** è la forma dei dati in ingresso e prende come parametri una shape di 4 dimensioni. In questo caso i valori sono:
    - Numero di timesteps che equivale a 39 e sta ad indicare il numero di frame per video sequenza.
    - Numero di features che equivale a 2 e sta ad indicare la quantità di righe della matrice di ogni frame
    - Lunghezza che equivale a 267 e sta ad indicare la quantità di colonne della matrice di ogni frame.
    - Numero dei canali che equivale ad 1 e sta ad indicare i canali che l'immagine possiede.
- **Activation** è il metodo di attivazione dello stato. In questo caso è stata impostata la funzione di attivazione ReLU.

ReLU è una funzione di attivazione ed è definita come la parte positiva del suo argomento dove z è l'input a un neurone. (Fig. 8)

$$f(z)=\max(0, z)$$

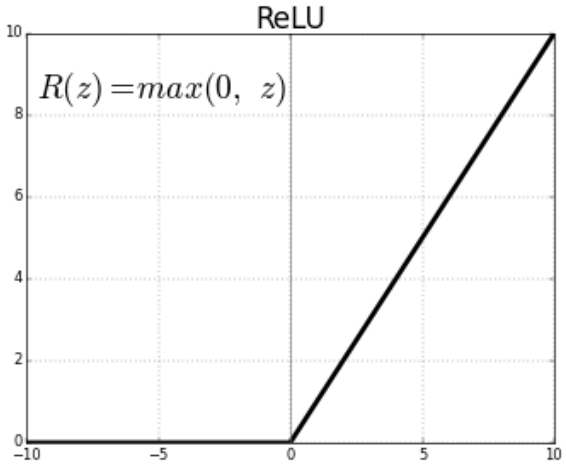

Fig. 8: funzione ReLU

È stata scelta la funzione ReLU poiché permette di ottenere un allenamento efficiente della rete in breve tempo ed è in grado di utilizzare insiemi di dati grandi e complessi.

Il secondo layer, Dropout, è un metodo di regolarizzazione che assegna randomicamente alle unità di input il valore 0 ad ogni step con una certa frequenza di rate presa come parametro durante la fase di allenamento, per far sì che il modello non vada in overfitting. Gli ingressi non impostati su 0 vengono aumentati di 1/(1 - rate) in modo che la somma di tutti gli ingressi rimanga invariata. È stato scelto questo layer poiché ci permette di prevenire efficacemente l'overfitting non modificando comunque la dimensione dell'input ricevuto. È stato inoltre impostata la frequenza di rate a 0.5 per avere un'efficacia maggiore ma comunque moderata, essendo i frame delle video sequenze di natura binaria.

Il terzo layer, Flatten, appiattisce l'input, fornendo in output dei dati di shape monodimensionale. Inoltre, non influisce sulla batch size. È stato utilizzato per rendere la shape dei dati

in output compatibile con i layer Dense e con la shape del file di test.

L'ultimo layer, Dense, è uno strato convoluzionale densamente connesso che aggiunge interessanti proprietà di non linearità, poiché può modellare qualsiasi funzione matematica. La funzione di output di questo layer è la seguente. (Fig. 9)

```
output = activation(dot(input, kernel) + bias)
```

Fig. 9: funzione output layer

- **Input:** rappresenta i dati ricevuti in input.
- **Kernel:** rappresenta il peso di questi dati.
- **Dot:** è una funzione che esegue il prodotto numpy sui vari input in base al loro peso.
- **Bias:** è un valore usato nel machine learning per l'ottimizzazione del modello.
- **Activation:** è la funzione di attivazione.

Il layer è stato utilizzato per estrapolare in modo non lineare le caratteristiche più importanti dai frame in input.
Nel primo caso è stata utilizzata la funzione di attivazione ReLU con un numero di unità in output pari a 100, mentre nel secondo caso è stata utilizzata la funzione softmax con un numero di unità in output pari al numero di emozioni considerate.

In matematica, una funzione softmax, è una generalizzazione di una funzione logistica che comprime un vettore k-dimensionale z di valori reali arbitrari in un vettore k-dimensionale f(z) di valori compresi in un intervallo (0,1) la cui somma è 1. Questa funzione è utilizzata per i metodi di classificazione multi-classe.

$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$$

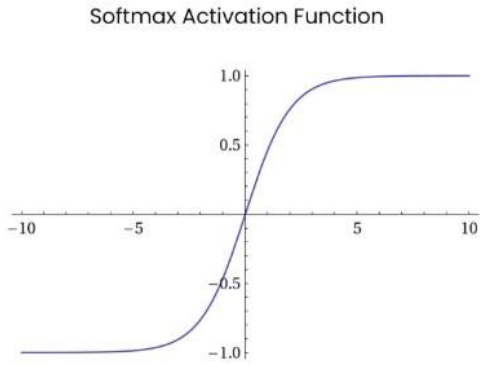

Fig. 10: funzione softmax

È stata utilizzata poiché il modello costruito deve eseguire una classificazione delle video sequenze in base alla classe di emozione e siccome il layer Dense che la utilizza fornisce in output una dimensione pari al numero di classi, questa funzione è la più adatta a questo tipo di elaborazione dei dati.

Una volta creato il modello, viene utilizzato compile per poterlo configurare prendendo, in questo caso, tre parametri:

- **Loss:** impostato al valore 'categorical_crossentropy'. È una funzione di loss utilizzata nelle attività di classificazione multi-classe ed è in grado di calcolare la differenza tra la distribuzione di probabilità fornita dall'ultimo strato Dense che deve utilizzare la funzione di attivazione softmax e quella definita dal file contenente la lista di classi corrispondenti al train e test. È necessario che l'ultimo strato della rete sia Dense con funzione di attivazione softmax perché esso dà in output una distribuzione di probabilità di dimensione pari al numero di classi da elaborare.

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

  È stata scelta questa funzione di loss per le sue particolari caratteristiche e poiché il nostro modello opera una classificazione multi-classe.

- **Optimizer:** impostato al valore 'adam'. L'ottimizzazione di Adam (deriva da adaptive moment estimation) è un metodo aleatorio che sfrutta la discesa del gradiente ovvero un algoritmo iterativo di ottimizzazione per trovare il minimo di una funzione. Adam ottimizza le iterazioni facendo convergere più velocemente la funzione. È stato scelto questo metodo di ottimizzazione poiché Adam è un algoritmo in grado di raggiungere rapidamente buoni risultati richiedendo poca memoria, infatti è uno dei più utilizzati in ambito deep learning.
- **Metrics:** impostato al valore accuracy. Esso indica che la metrica che il modello appena compilato vuole calcolare è l'accuratezza. È stato scelto questo valore in modo che il modello fosse in grado di calcolare e fornirci la metrica dell'accuratezza.

V. ADDESTRAMENTO RETE NEURALE

Durante l'addestramento della rete neurale sono state testate varie configurazioni sia del dataset che della rete stessa per giungere al risultato ottimale.

A. *Fase 1*

Per prima cosa, la rete è stata addestrata con il dataset costruito in precedenza, aggiungendo dei frame di padding per fare in modo che ogni video avesse lo stesso numero di frame e che quindi il dataset fosse accettato dalla rete.
Per costruire il padding si è preso come limite superiore il video con più frame, in particolare ne aveva 39, e si è fatto in modo che tutti gli altri video ne avessero lo stesso numero aggiungendo alla destra e alla sinistra nuovi frame con tutti i valori a 0 fino a raggiungere i 39 frame. Inoltre, ogni frame letto è stato elaborato sostituendo i valori dei pixel 255 con il valore 1, in modo da rendere binarie le matrici risultanti.

Tutti i video processati sono stati inseriti in un array numpy, insieme ai loro label, che verranno poi utilizzati per creare i file di train e test.

```
rightLabels = ['1', '2', '3', '4', '5', '6', '7']
labelsTrain = ['0', '1', '2', '3', '4', '5', '6']
def get_data(data_dir):
    data = []
    l=[]
    for label in labelsTrain:
        lbl=rightLabels[labelsTrain.index(label)]
        path = os.path.join(data_dir, lbl)
        subs= next(os.walk(path))[1]
        for sub in subs:
            imgs=next(os.walk(path+"/"+sub))[2]
            temp=[]
            lenght= len(imgs)
            resto= 39-lenght
            if resto==0:
                parte1=0
                parte2=0
            elif (resto%2)==0:
                parte1=resto/2
                parte2=resto/2
            else:
                parte1=(resto/2)+0.5
                parte2=(resto/2)-0.5
            for n in range(int(parte1)):
                temp.append(np.zeros((267,267)))
            for img in imgs:
                img_arr = cv2.imread(os.path.join(path+"/"+sub, img), 0)
                img_arr= cv2.resize(img_arr, (267,267))
                threshold =0.2
                img_arr= np.where(img_arr/255>=threshold, 1, 0)
            for n in range(int(parte2)):
                temp.append(np.zeros((267,267)))
            l.append(label)
            data.append(np.asarray(temp))
    return np.asarray(data), to_categorical(np.asarray(l).astype(np.float32))
```

Fig. 11: padding

Sono stati a questo punto creati i file di train e test utilizzando i dati processati in precedenza, come si evince dalla figura sottostante. I file sono stati divisi in 70% train e 30% test.

```
x, y = get_data('test1_dataset/')
trainx, testx, trainy, testy = train_test_split(x, y,
test_size=0.30, random_state=1, shuffle=True)
```

Fig. 12: split train e test

Quando si parla di machine learning si parla spesso di train, questo perché i vari modelli osservano i dati che vengono passati imparandone pattern e ricorrenze. Utilizzando il file di train il nostro modello imparerà le relazioni tra le nostre X (variabili di input) e Y (ovvero il target, l'output). In questa fase i modelli osservano, studiano, analizzano e cercano di fare previsioni su quanto imparato. Il modello può così confrontare il risultato della sua predizione con quello reale e di conseguenza aggiornare i vari parametri per ridurre al minino l'errore. Infine, viene utilizzato il file di test per validare il nostro modello.

È stata costruita a questo punto la rete in cui inserire i file appena generati per l'addestramento.
La rete, come già descritto, utilizza il layer ConvLSTM2D, poi uno strato di Dropout, uno strato di Flatten per rendere la matrice 1D, e due strati di Dense di cui il secondo con parametro il numero di classi. Viene infine eseguita la compilazione del modello ed il fit e poi vengono stampati il classification report, la matrice di confusione e i grafici di accuracy e loss.

La loss misura l'errore nell'addestramento e quindi la quantità di informazione perse, perciò l'obiettivo principale è quello di minimizzare questa funzione attraverso varie fasi di ottimizzazione.

Nel campo del machine learning e nella classificazione statica, la matrice di confusione è una matrice che rappresenta le prestazioni di un algoritmo di apprendimento. Essa è una matrice quadrata che rileva il conteggio dei veri positivi e dei veri negativi, dei falsi positivi e dei falsi negativi nelle previsioni di un classificatore. Da questa matrice, quindi, è stato possibile comprendere le performance dell'applicazione sviluppata in modo da determinare quanto il modello sviluppato sia stato accurato ed efficace.

Il classification report utilizza le seguenti metriche base della matrice di confusione:

- **Accuracy:** indica l'accuratezza del modello e viene utilizzata per valutare i modelli di classificazione. Informalmente è la frazione di previsioni che il nostro modello ha ottenuto correttamente. La migliore accuratezza è 1, mentre la peggiore è 0. Formalmente l'accuratezza si definisce così:

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

- **Recall:** detta anche sentivity 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.

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

- **F-score (o punteggio f):** è una media armonica ponderata delle metriche Precision e Recall in modo tale che il punteggio migliore sia 1 e il peggiore sia 0. La media ponderata di F1 è utilizzata per confrontare i modelli di classificatore.

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

- **Precision:** 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.

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

- **Support:** indica il numero di samples per ogni classe del file di test.
- Le medie riportate includono **macro average** (macro-avg) e **weighted average** (weighted avg). La macro avg calcola la media non ponderata per labels, mentre weighted avg calcola la media ponderata in base al numero di support per label.

Purtroppo, l'addestramento del modello descritto in precedenza non è andato a buon fine a causa della grandezza

del dataset, infatti si parla di 39 frame a video compreso il padding, di cui ognuno aveva dimensione 267x267 pixels e la rete non è riuscita a sopportare il carico dell'addestramento dando errore di out of memory (memoria insufficiente).

B. *Fase 2*

A causa quindi dell'errore di out of memory è stato deciso di semplificare il dataset.
Per fare ciò si è deciso di utilizzare una strategia particolare: fare la media ogni quattro pixel di ogni frame e produrne soltanto uno. (fig. 13A)

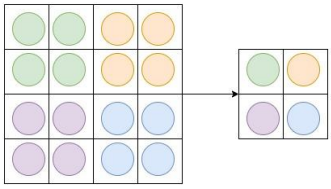

Fig. 13A: riduzione frame

In questo modo vengono dimezzate le righe e colonne di ogni frame. Infatti, partendo dalla dimensione dei frame di 267x267px riusciamo ad ottenere un nuovo frame di dimensione 134x134px. Purtroppo, anche in questo caso la rete non è riuscita a sostenere il carico di lavoro dando di nuovo errore di out of memory. Si è pensato quindi di ridurre ulteriormente il frame, riapplicando la stessa strategia e portandolo da una dimensione di 134x134px ad una dimensione di 67x67px. Anche con questa dimensione molto ridotta però l'addestramento non è andato a buon fine. Dopo vari test si è giunti alla conclusione che bisognava effettuare più volte questa riduzione in modo che ogni frame avesse una dimensione minore o uguale a 34x34 pixels. (fig. 13B)

```
def reduceImage(arr, rows, columns):
    if (rows%2)!=0:
        r=int((rows-1)/2)
        newDR= int((rows+1)/2)
    else:
        r=int(rows/2)
        newDR=int(rows/2)
    if (columns%2)!=0:
        c=int((columns-1)/2)
        newDC= int((columns+1)/2)
    else:
        c=int(columns/2)
        newDC= int(columns/2)
    x,y, i, j=0,0,0,0
    b=np.zeros((newDR,newDC))
    for _ in range(r):
        for _ in range(c):
            if (j+1)<rows and (i+1)<columns:
                b[x][y]=np.mean([arr[i][j], arr[i][j+1], arr[i+1][j], arr[i+1][j+1]])
            y+=1
            j+=2
        x+=1
        y=0
        i+=2
        j=0

    if (arr.shape[0]%2)!=0:
        x, i=0, 0
        for _ in range(r):
            b[i][c-1]=np.mean([arr[x][columns-1], arr[x+1][columns-1]])
            x+=2
            i+=1
        b[r-1][c-1]=arr[rows-1][columns-1]
    if(arr.shape[1]%2)!=0:
        x, i=0, 0
        for _ in range(c):
            b[r-1][i]=np.mean([arr[rows-1][x], arr[rows-1][x+1]])
            x+=2
            i+=1
        b[r-1][c-1]=arr[rows-1][columns-1]
```

Fig. 13B: codice riduzione frame

Questa soluzione ha finalmente permesso alla rete di riuscire ad elaborare i dati, ma allo stesso tempo l'utilizzo multiplo di questa strategia che ha portato a ridurre in maniera importante le dimensioni del frame, ha anche costretto ad effettuare più volte la media dei pixel adiacenti, portando ad una grande perdita di informazioni in confronto al frame originale. Tutto ciò ha quindi condotto ad un cattivo addestramento della rete, con un'accuracy molto bassa e una loss molto alta

C. *Fase 3*

Si è pensato a questo punto di fare una diversa riduzione delle dimensioni dei frame. È stata creata quindi una matrice 2x267px dove ogni elemento i-esimo della prima riga fosse ricavato dalla media di tutti i valori della colonna i-esima ed ogni elemento j-esimo della seconda riga fosse ricavato dalla media di tutti i valori della riga j-esima. (fig. 14)

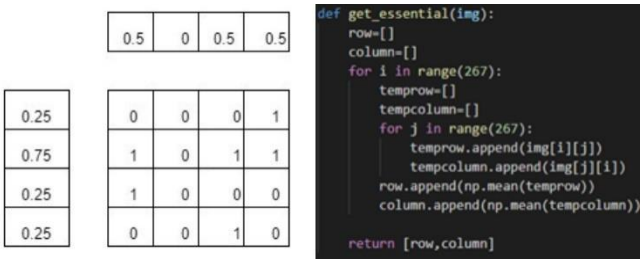

Fig. 14: Riduzione frame 2x267

Utilizzando una matrice costruita in questo modo si riesce ad avere una buona propagazione delle zone di interesse, conservando quindi le caratteristiche del frame. Con questi risultati siamo riusciti ad allenare la stessa rete precedente efficientemente. (Fig.15, Fig. 16, Fig. 17, Fig. 18)

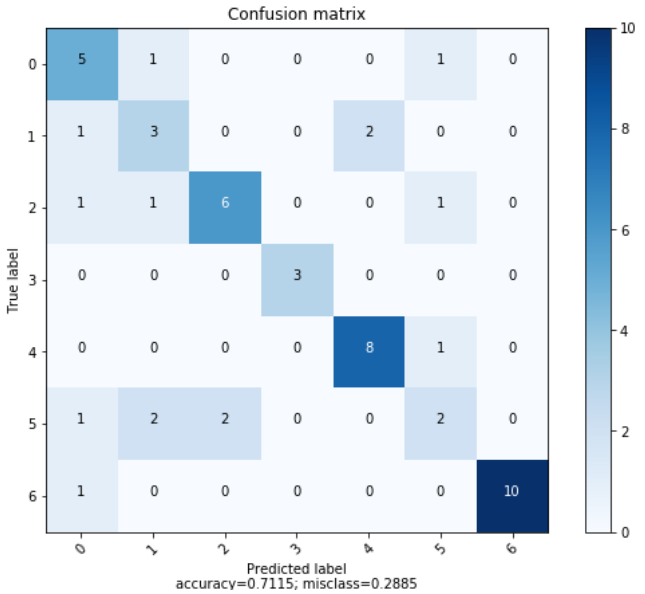

Fig. 15: Matrice di confusione

Classification Report

	precision	recall	f1-score	support
0	0.56	0.71	0.63	7
1	0.43	0.50	0.46	6
2	0.75	0.67	0.71	9
3	1.00	1.00	1.00	3
4	0.80	0.89	0.84	9
5	0.40	0.29	0.33	7
6	1.00	0.91	0.95	11
accuracy			0.71	52
macro avg	0.70	0.71	0.70	52
weighted avg	0.72	0.71	0.71	52

Fig. 16: Classification report

Fig. 17: grafico accuracy

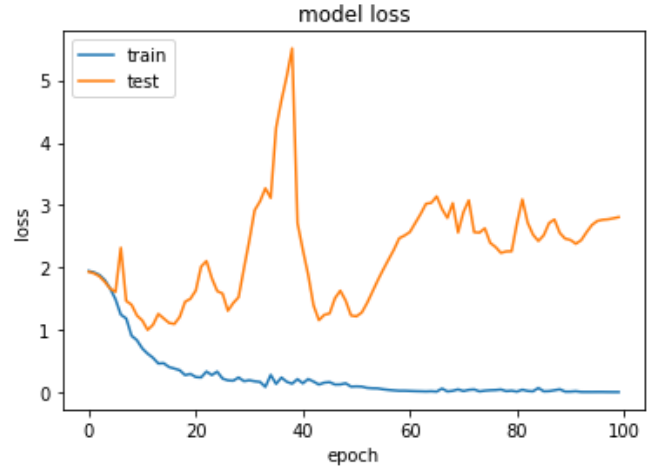

Fig. 18: grafico loss

La matrice di confusione e il classification report mostrano un buon risultato per l'addestramento, garantito anche dal grafico dell'accuracy che raggiunge un valore del 71% circa. Il problema principale però è la validation loss che, come si evince dal grafico, è molto altalenante con valori che tendono ad incrementare col passare delle epoche.

Si è subito capito di essere in una situazione di overfitting della rete.
L'overfitting è un fenomeno che avviene quando un modello statistico molto complesso si adatta ai dati osservati, cioè tende a memorizzarli, perché ha un numero eccessivo di parametri da considerare rispetto al numero di osservazioni fornitegli.

D. *Fase 4*

La prima ipotesi è stata quella di aggiungere uno o più strati di Dropout in modo da prevenire questo fenomeno, ma questo non ha fatto altro che peggiorare la situazione, poiché si riduceva maggiormente l'input diverso da 0 alla rete. Era infatti già sufficiente il singolo layer di Dropout inserito nel modello. Dopo vari test, si è giunti alla conclusione che il problema non fosse il modello, ma il fatto che a causa del dover bilanciare i video per emozioni, si era ridotto di molto il numero di quest'ultimi, avendone pochi per alcune emozioni.

A questo punto per risolvere il problema si è pensato di eliminare alcune emozioni dal nostro dataset, quelle con meno video che avevano costretto ad un bilanciamento a ribasso. In questo modo, si potevano utilizzare molti più video per emozione.

Sono state quindi eliminate le emozioni 2, 4 e 6, rispettivamente disprezzo, paura e tristezza lasciando le emozioni 1, 3, 5 e 7 che sono rabbia, disgusto, felicità e sorpresa. In questo modo, a seguito di un nuovo bilanciamento, si è riusciti ad ottenere un valore di 35 video per emozione di cui ognuno di 39 frame.

Il modello non ha subito cambiamenti da quello precedente e l'addestramento è andato a buon fine. (Fig.19, Fig.20, Fig.21, Fig.22)

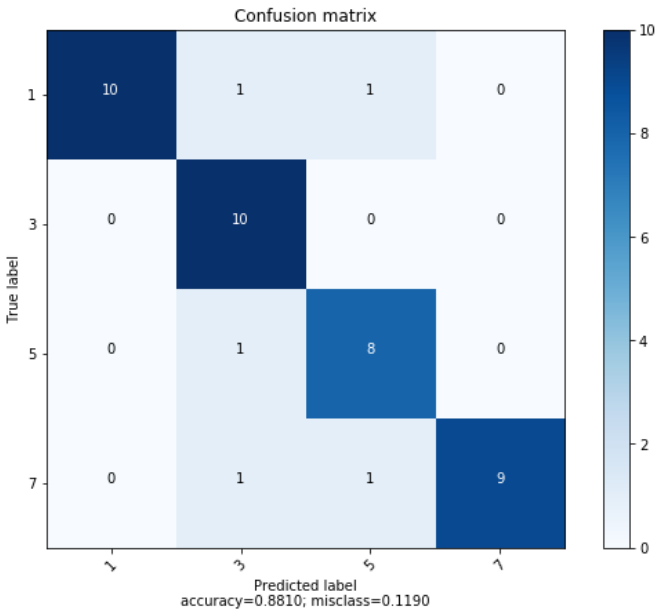

Fig. 19: matrice di confusione

Classification Report

	precision	recall	f1-score	support
1	1.00	0.83	0.91	12
3	0.77	1.00	0.87	10
5	0.80	0.89	0.84	9
7	1.00	0.82	0.90	11
accuracy			0.88	42
macro avg	0.89	0.89	0.88	42
weighted avg	0.90	0.88	0.88	42

Fig. 20: classification report

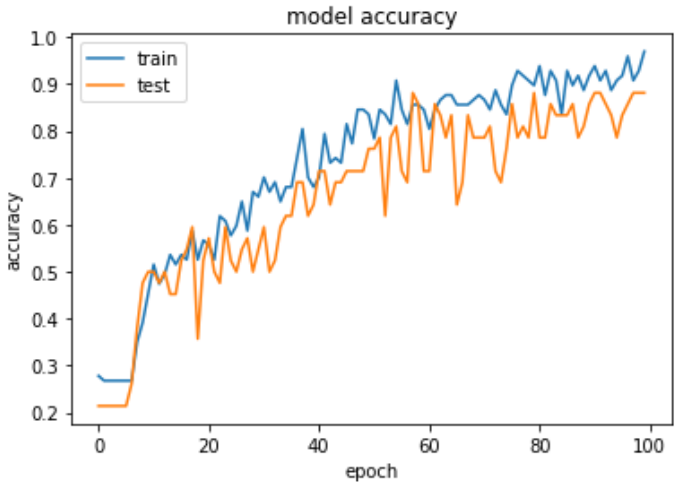

Fig. 21: grafico accuracy

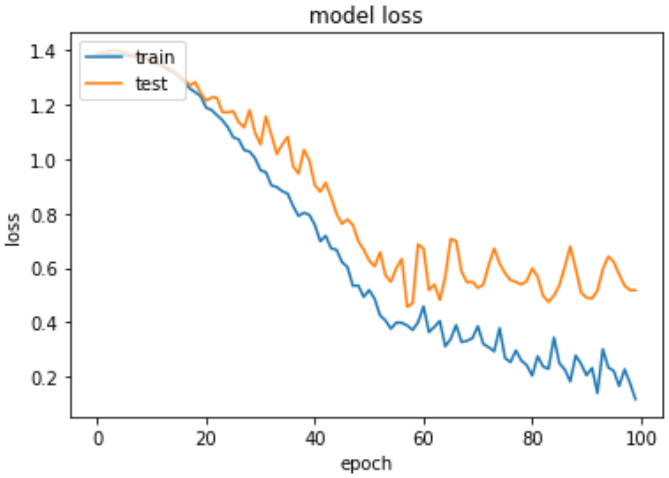

Fig. 22: grafico loss

Come si evince dalla matrice di confusione, per ogni label la rete ha riconosciuto correttamente la maggior parte delle video sequenze, con un f1-score di circa l'88% in media.

Dai valori ottenuti si evince un miglioramento, anche se lieve in termini di accuracy, passando da un valore del 71% nella fase precedente a un valore dell'88%. Un importante miglioramento è quello però della validation loss, che adesso assume un andamento più stabile tendente allo 0 con l'aumentare delle epoche.

VI. CONCLUSIONI

In conclusione, il modello finale ha raggiunto ottimi risultati, essendo in grado di predire efficientemente circa l'88% delle video sequenze fornitegli, classificandole con le emozioni corrette.
Inoltre, nonostante ci si è trovati davanti a diverse difficoltà durante la fase di sviluppo della rete, esse sono state superate agilmente rielaborando il dataset, in modo che l'addestramento andasse a buon fine e che i risultati della rete migliorassero.
Alcune di queste difficoltà sono state l'out of memory, l'addestramento randomico della rete e l'overfitting. Si è appreso essere problemi molto comuni in quest'ambito.
L'out of memory può essere superato elaborando al meglio il dataset fornito, estraendone quindi solo le caratteristiche importanti e riducendone il carico da affidare alla rete. Nel nostro caso si è risolto riducendo le dimensioni dei vari frame delle video sequenze da una matrice 267x267 ad una 34x34.
L'addestramento randomico della rete è causato da una cattiva elaborazione del dataset, da cui non si riescono ad estrarre le caratteristiche fondamentali per un corretto training. Il modo migliore per risolvere questo problema è quello di cercare di capire se il dataset così elaborato conserva le informazioni essenziali o se ha bisogno di essere processato diversamente. Nel nostro caso si è capito che una riduzione dei frame del dataset ad una dimensione di 34x34px attraverso il processo spiegato nella fase 2 del progetto portava con sé una perdita di informazioni del frame e che quindi bisognava cambiarne il metodo di elaborazione. Per farlo si è pensato di costruire una matrice 2x267 come spiegato nella fase 3 del progetto.
L'overfitting invece può essere superato in svariati modi. Si potrebbe per prima cosa aggiungere un layer di Dropout al modello della rete, in modo da prevenirlo. Un'altra importante tecnica è quella di bilanciare correttamente il dataset fornito alla rete ed assicurarsi di avere abbastanza dati in input per un corretto allenamento. Altre tecniche riguardano come in precedenza un'elaborazione migliore del dataset per estrarre al meglio le caratteristiche più importanti. Nel nostro caso si è risolto eliminando alcune emozioni che possedevano pochi video e che costringevano a causa del bilanciamento a ridurne la quantità anche per le altre.
Questo progetto ha portato alla luce nuove conoscenze sia in ambito Python che in quello del machine learning applicato ad una rete neurale.
Sarebbe interessante lavorare su sviluppi futuri, utilizzando la rete addestrata in vari campi applicativi, come quelli della robotica, del gaming e molti altri.

VII. SITOGRAFIA

[1] Keras, https://keras.io/api
[2] Wikipedia, https://it.wikipedia.org/wiki
[3] Medium, https://medium.com
[4] Stackoverflow, https://stackoverflow.com
[5] Towards data science, https://towardsdatascience.com