**RELAZIONE CASO DI STUDIO**

**ESAME DI**

**SISTEMI AD AGENTI**

**ANNO ACCADEMICO 2019/2020**

**Studenti:**

**Celenza Francesco 680053**

**Lopane Nicola 688153**

# Obiettivo dello Studio

L’obiettivo prefissato per questo studio è effettuare un confronto tra il modello VGGFace (basato su VGG16) ed il modello VGGFace (basato su ResNet50). Nello specifico, per entrambi i modelli verrà eseguito il training su un numero consistente di epoche e successivamente verrà applicato un approccio di training denominato “**Dense-Sparse-Dense” (DSD).**  Con i risultati che si otterranno verrà effettuato un confronto tra la prima fase di training e la fase DSD per verificare le migliorie che quest’ultima apporterà ai modelli (qualora ci dovessero essere).

# VGGFace

## Introduzione al modello

VGGFace è un modello utilizzato per il **Face Recognition**, pre-addestrato su un dataset di milioni di immagini contenenti volti(chiamato VGGface dataset). Il dataset è uno dei più usati per sviluppare **Reti Neurali Convoluzionali (CNN)** per gli obiettivi di Face Recognition come **identificazione** e **verifica** facciale. In questo caso di studio si sono considerati due modelli di VGGFace, il primo basato sul modello VGG16 (Una CNN con 16 layers), il secondo invece basato sul modello di rete ResNet50 (una Residual Network con 50 layers).

## Training del modello

Per il training dei modelli, si è seguito il paper denominato “**Local Learning With Deep and Handcrafted Features for Facial Expression Recognition**” (<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8716652>)**,** redatto da tre professori dell’Università di Bucarest, M. I. Georgescu, R. T. Ionescu e M. Popescu. Essi addestrano vari modelli, con vari dataset, e ne confrontano i risultati per verificare quale esibisce una bontà migliore (per bontà si intende la capacità di un modello di saper identificare dei volti in base a delle emozioni).

In questo studio ci siamo concentrati sull’approccio utilizzato per il modello VGGFace (basato sul modello VGG16), il dataset utilizzato è **FER2013,** nel quale sono contenute 28709 immagini destinate al training, 3589 immagini per il Validation e, infine, altre 3589 immagini utilizzate per il Test. Tutte le immagini sono divise in 7 classi di emozioni, ovvero: **angry** (rabbia), **disgust** (disgusto)**, fear** (paura)**, happy** (felicità)**, normal** (neutralità)**, sad** (tristezza)e **surprise** (sorpresa). Il modello è composto da 16 layers dei quali sono stati congelati solo quelli convoluzionali, mentre sono stati addestrati i layers fully-connected. E’ stato effettuato il fine-tuning del modello applicando un approccio **Dense-Sparse-Dense (DSD),** il quale consiste nel suddividere l’addestramento in due fasi Dense e in una fase Sparse, utili per prevenire un eventuale overfitting del modello. Successivamente, al termine dell’addestramento, sono state aggiunte una seconda fase Sparse e un’ultima fase Dense.

Per quanto riguarda il modello VGGFace basato su ResNet50, si è eseguito lo stesso approccio, congelando i layers predefiniti della rete ed aggiungendo dei layer Dense, Dropout, Dense.

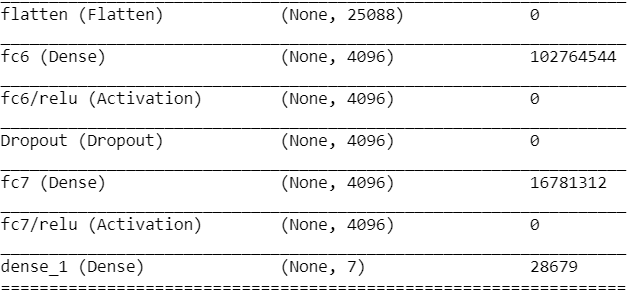
Nei paragrafi 2.3 e 2.4 è possibile trovare una descrizione dettagliata dell’approccio seguito.

### Image\_Classification per classificare FER2013

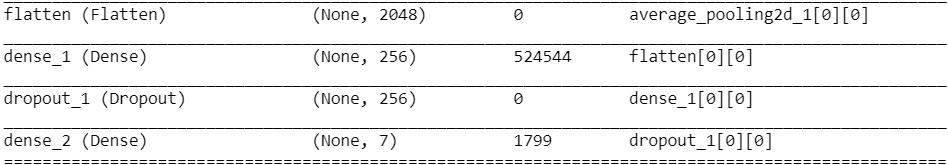
Prima di passare alla descrizione delle fasi di addestramento del modello, è opportuno specificare che nel dataset **FER2013**, le immagini sono tutte contenute in una tabella salvata in un file .csv. Tale tabella è composta da tre colonne: “Usage” specifica l’utilizzo di ogni immagine (se Training, Validation o Test), “Pixels” contiene i pixels di ogni immagine ed infine “Emotions” nella quale sono contenuti dei numeri interi compresi tra 1 e 6, utilizzati per specificare l’emozione di riferimento (1=angry, 2=disgust, 3=fear, 4=happy, 5=normal, 6=sad, 7=surprise). Per questo motivo, si è dovuto scrivere un programma che sia in grado di convertire i pixel in immagini e, in base ai valori di Usage e Emotion, catalogarle nella giusta cartella. Il programma quindi crea tre cartelle denominate “Train”, “Test” e “Val”, all’interno delle quali vengono inserite altre 7 cartelle ognuna corrispondente ad una specifica classe di emozione. Il file è stato denominato **Image\_Classification.**

## Fasi Dense

Innanzitutto è stato rimpiazzato l’ultimo layer softmax, formato da 1000 unità, con un layer softmax con tante unità quante sono le classi di emozioni(indicato con “dense\_1” nella figura 1.1) e successivamente è stato aggiunto un Dropout layer, dopo il primo layer fully-connected (denominato fc6/relu), con dropout rate impostato a 0.7. Nella figura 1.2 è possibile, invece, osservare i layers Flatten, Dense, Dropout, Danse aggiunti al modello ResNet50 allo scopo di ridurre l’overfitting.



*Figura 1.1: layers fully connected per il modello VGGFace.*

**

*Figura 1.2: layers Flatten, Dense, Dropout, Dense per il modello ResNet50.*

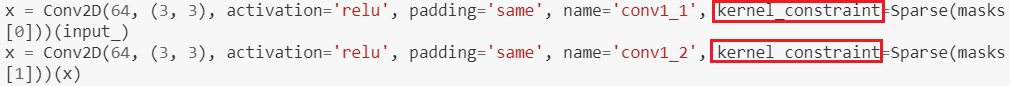
A questo punto sono state caricate le immagini contenute nelle cartelle di Train e Val e attraverso il metodo **ImageDataGenerator** sono state effettuate delle operazioni di preprocessing delle immagini, tra le quali anche un flip orizzontale. Successivamente attraverso il metodo **flow\_from\_directory()** sono stati creati i batch con le immagini preprocessate, è stato quindi impostato un valore di batch\_size per il train pari a 32, mentre per il validation, un valore pari a 8. Nel paper è stato impostato un valore di learning rate di **1e-4** ed è stato impostato in maniera tale da essere decrementato di un fattore di 10 nel caso in cui la val\_loss non sarebbe migliorata per più di 10 epoche. Per seguire tale approccio, si è utilizzata la funzione **ReduceLROnPlateau.**

Infine è stata utilizzata la funzione **CSVLogger** per creare un file di log contenente tutte le informazioni del training per ogni epoca.

E’ opportuno specificare che nell’addestramento sono presenti 3 fasi Dense, quanto descritto finora viene applicato solo alla prima fase Dense, mentre nelle altre due fasi vengono utilizzati gli stessi layer della prima fase, usando però gli stessi pesi del modello addestrato nella fase precedente. Infatti, per ogni fase (sia Dense che Sparse), dopo il training, viene salvato il modello con tutti i pesi relativi ai layers. Nella fase successiva, questi pesi vengono caricati attraverso il metodo **load\_weights()**, e impostati per il nuovo addestramento. Il numero di epoche impostato per la prima fase Dense è 200, mentre per le altre fasi Dense è impostato a 50. Infine, nella seconda fase Dense, il learning rate è uguale a quello ottenuto nella prima fase Sparse(presente dopo la prima fase Dense), mentre nella terza fase Dense, il learning rate è uguale a quello ottenuto nella seconda fase Sparse(presente dopo la seconda fase Dense).

## Fasi Sparse

Una fase Sparse viene utilizzata per evitare l’overfitting di un modello. L’impostazione è analoga a quella descritta per la fase Dense, quindi sia nella prima fase Sparse che nella seconda vengono utilizzati gli stessi layer e vengono caricati i pesi del modello addestrato nella fase precedente. Successivamente è stata regolarizzata la rete rimuovendo le connessioni con pesi piccoli, per ogni layer viene calcolata una percentuale del peso e se essa dovesse essere vicino a 0 (in valore assoluto), viene impostato a 0 il valore di quel peso. Questo è stato possibile grazie ad una **sparsity\_mask**, nella quale è stato impostato uno **sparsity\_rate** del **60%**, con il quale è stato possibile calcolare la percentuale di ogni peso. E’ opportuno precisare che è stato attuato questo procedimento solo per i layer convoluzionali, andando ad impostare un **kernel\_constraint**(rettangoli rossi in figura 1.3).

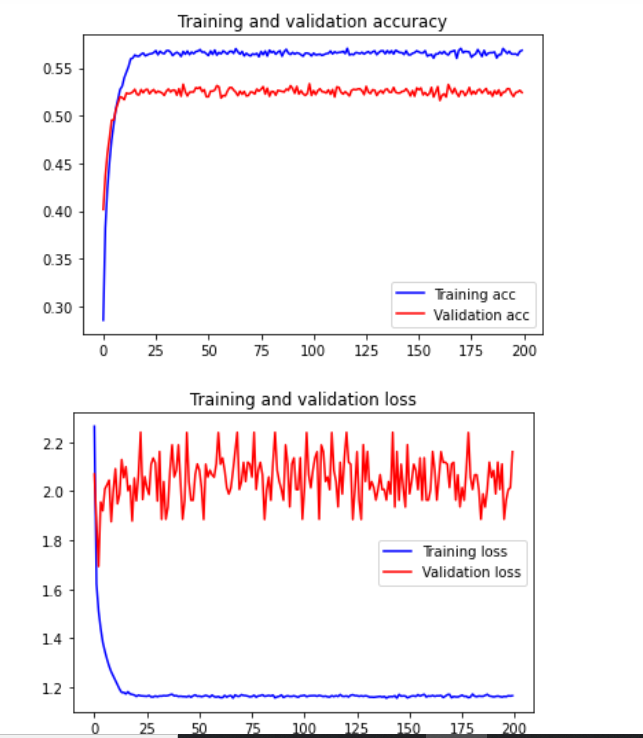
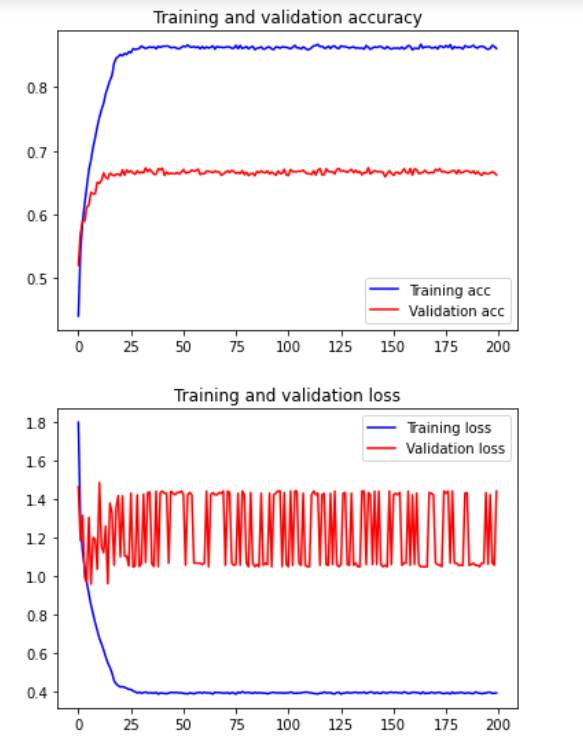
*Figura 1.3: Esempi di layer convoluzionali con aggiunta di un kernel\_constraint.*

Come specificato nella fase Dense, anche per le fasi Sparse il numero di epoche impostato è 50. Il valore del learning rate nella prima fase Sparse è uguale al valore ottenuto dalla fase precedente (prima fase Dense), mentre il learning rate della seconda fase Sparse è uguale al valore ottenuto nella seconda fase Dense.

# Risultati

## Prima fase Dense

Come anticipato nei paragrafi precedenti, nella prima fase Dense i modelli sono stati addestrati per **200** epoche e con un learning rate pari a **1e-4**. Di seguito è possibile osservare i risultati ottenuti.

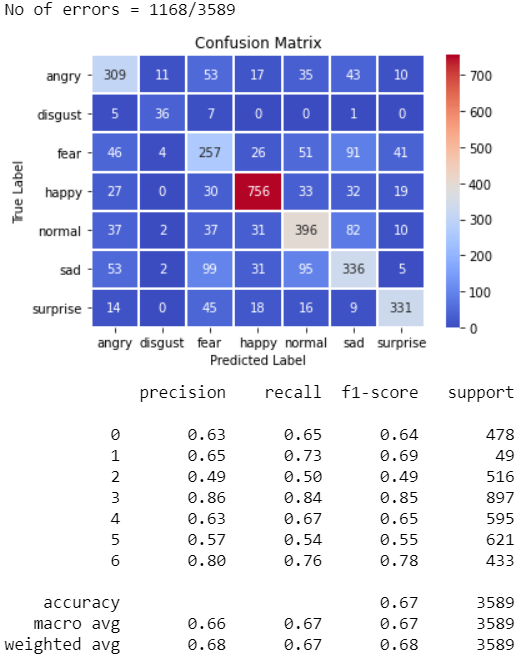
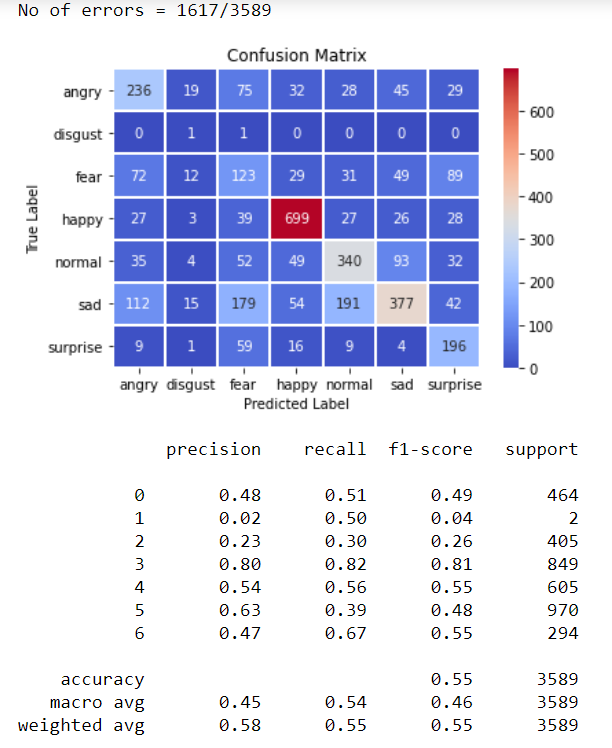
**

*Risultati relativi al modello VGGFace Risultati relativi al modello ResNet50.*

Per quanto riguarda il modello VGGFace basato su vgg16, si è ottenuto un valore di accuracy sul training pari a **0.861**, mentre l’accuracy sul validation è stata di **0.6626.** La loss sul training è stata di **0.395**, mentre quella sul validation è stata di **1.44**.

Per il modello ResNet50 invece, il valore di accuracy sul training è stato di **0.569,** mentre sul validation il valore è stato di **0.5244**. Infine si è ottenuto un valore di loss sul training di **1.165** mentre sul validation il valore è stato di **2.16.**

### Test dei modelli

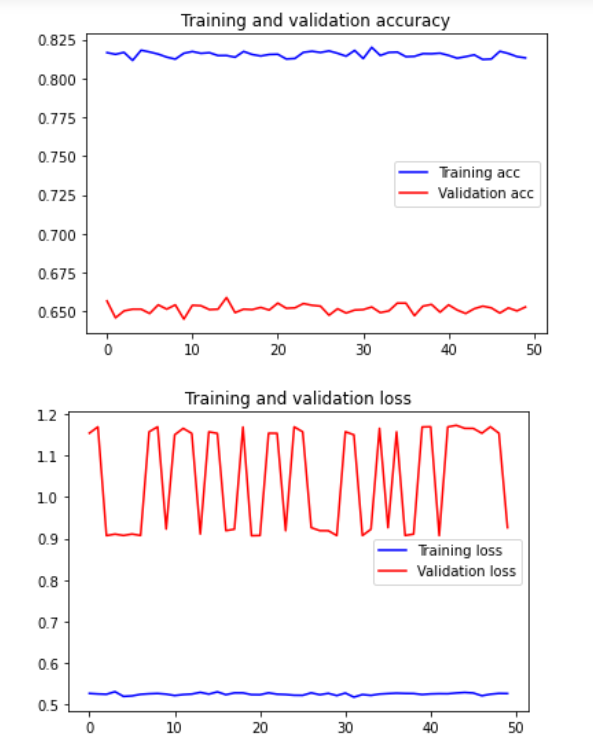
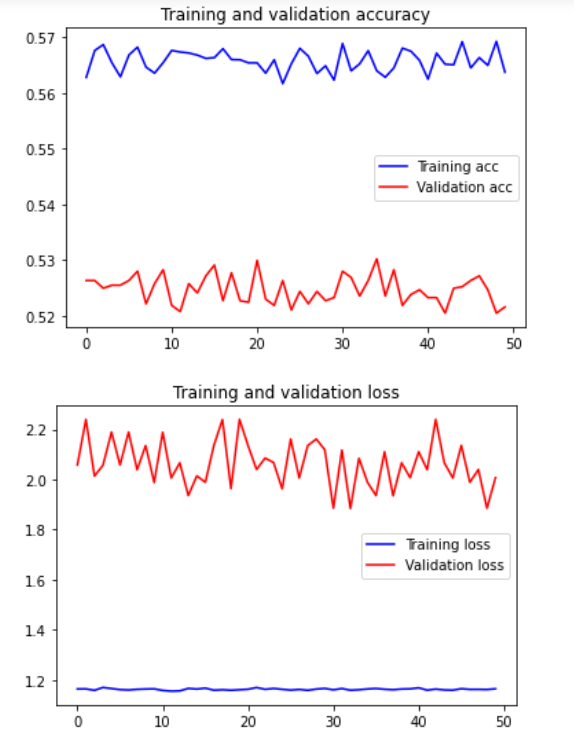
Dopo l’addestramento è stato eseguito il test dei modelli su un numero di immagini pari a 3589.

Matrice di confusione VGGface. Matrice di confusione ResNet50

Dalle matrici di confusione si evince come la classe di emozione **happy,** è stata quella classificata con maggiore successo (con un valore di f1 pari a **0.85** per la VGGFace e di **0.81** per ResNet50). Al contrario invece, nel caso della VGGFace la classe che ha ottenuto una classificazione peggiore è stata **fear** (**0.49**)**,** mentre per il modello ResNet50 è stata **disgust** (**0.04**)**.**

## Prima fase Sparse

Nella prima fase sparse i modelli sono stati addestrati per 50 epoche. Il valore di learning rate ottenuto dalla prima fase Dense, per la VGGFace è stato di **1e-23**, stesso valore si è ottenuto anche per il modello ResNet50. Di seguito è possibile osservare i risultati ottenuti.



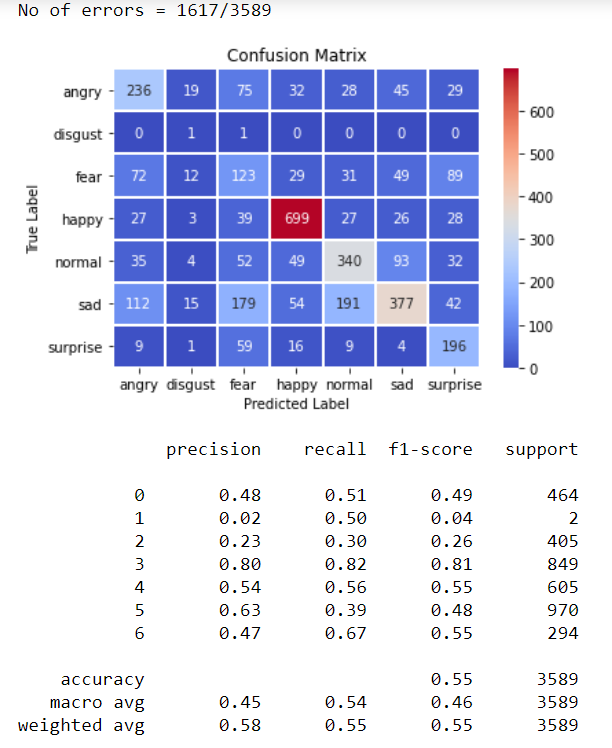
Risultati relativi al modello VGGFace Risultati relativi al modello ResNet50

Nella prima fase Sparse, per la VGGFace si è ottenuto un valore di accuracy sul training pari a **0.813**, mentre l’accuracy sul validation è stata di **0.6528.** La loss sul training è stata di **0.5267**, mentre quella sul validation è stata di **0.926**.

Per quanto riguarda il modello ResNet50, il valore di accuracy sul training è di **0.564**, mentre sul validation è di **0.522**. La loss sul training è stata di **1.165** mentre sul validation **2.007**.

Comparando i risultati ottenuti in questa fase, con la fase precedente, si può notare come siano leggermente peggiorati i valori di accuracy sul training. Questo avviene perchè la fase Sparse è utilizzata per cercare di ridurre l’overfitting di un modello, riducendo quindi la loss sul validation. Questo, infatti, ha comportato una riduzione di loss per la VGGFace, da un valore di 1.44 a 0.926. Si è ottenuta la stessa situazione anche per quanto riguarda il modello ResNet50, la cui loss è passata da 2.16 a 2.007.

### Test dei modelli

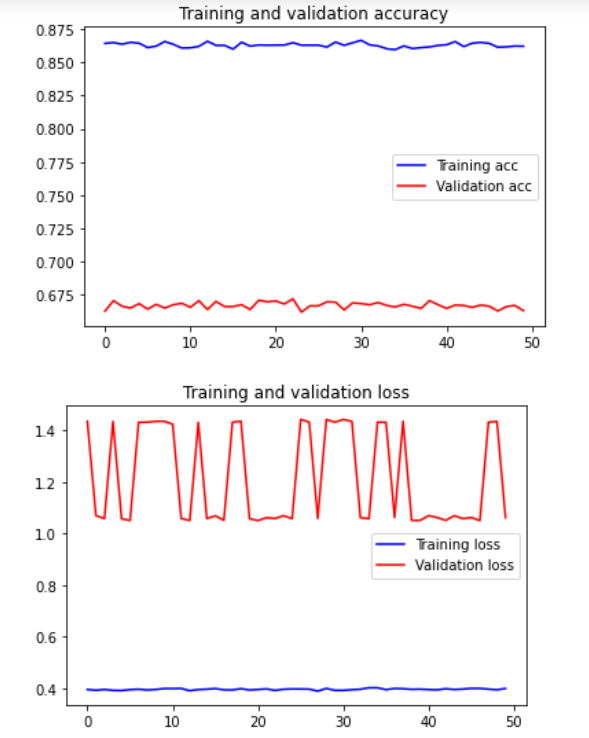
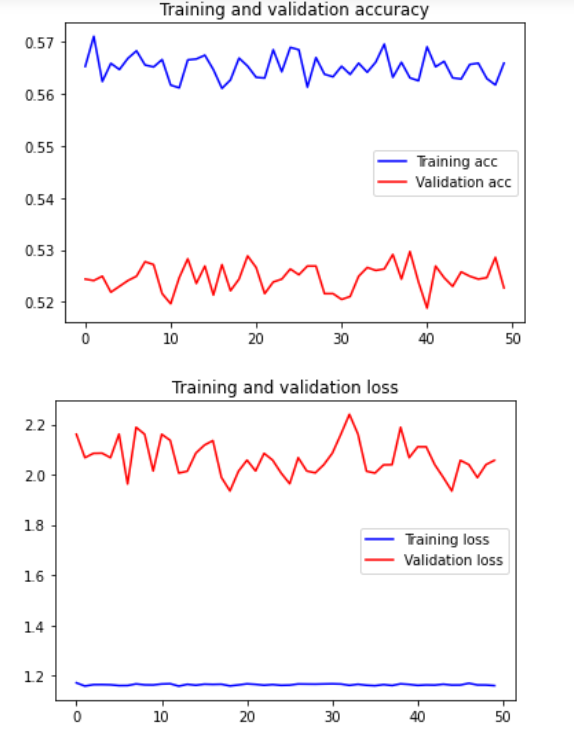


Matrice di confusione VGGFace Matrice di confusione ResNet50

Dalle matrici di confusione è possibile osservare come si sia ottenuta la stessa situazione della fase precedente. E’ opportuno specificare, infatti, che si otterrà tale situazione anche per tutte le altre fasi, per questo motivo si è deciso di omettere l’inserimento dei test delle altre fasi.

## Seconda fase Dense

Nella seconda fase Dense i modelli sono stati addestrati per altre 50 epoche. Nel caso della VGGFace il valore di learning rate ottenuto dalla fase precedente è **1e-26**, mentre per il modello ResNet50 è stato **1e-25.**

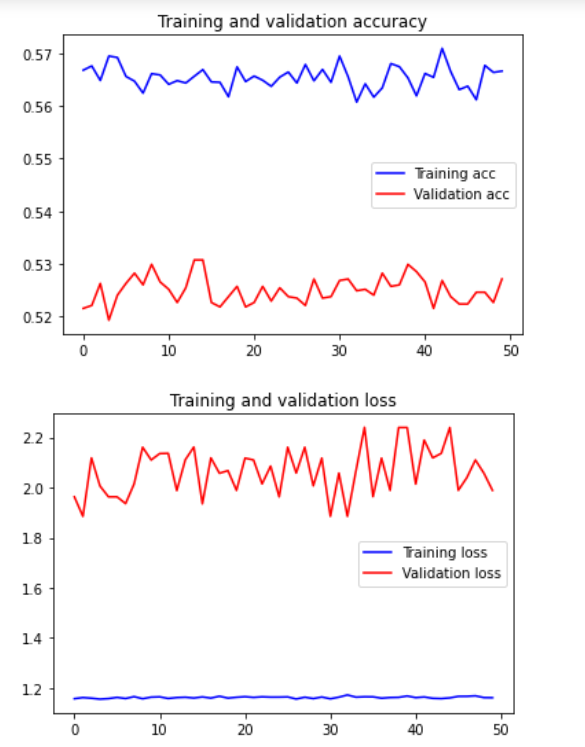
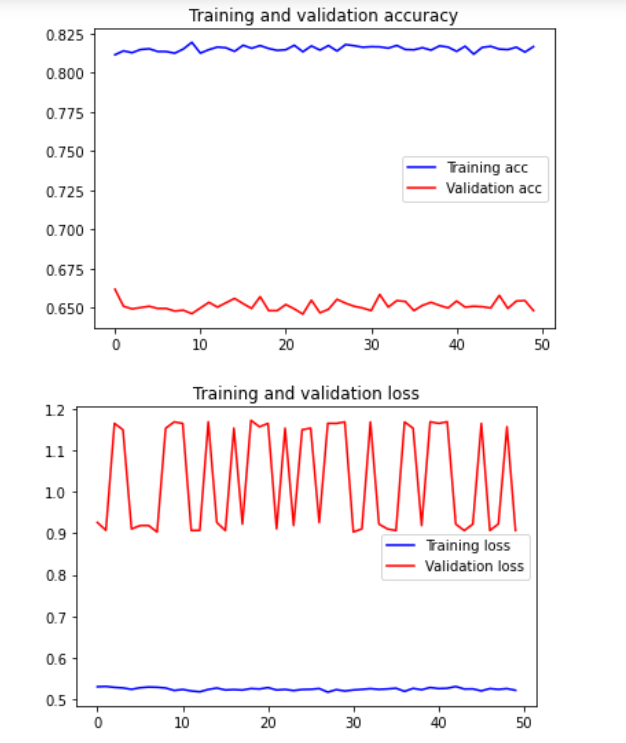
  
 Risultati relativi al modello VGGFace Risultati relativi al modello ResNet50

Nella seconda fase Dense, per la VGGFace si è ottenuto un valore di accuracy sul training pari a **0.862**, mentre l’accuracy sul validation è stata di **0.6656.** La loss sul training è stata di **0.396**, mentre quella sul validation è stata di **1.425**.

Per quanto riguarda il modello ResNet50, il valore di accuracy sul training è di **0.566**, mentre sul validation è di **0.523**. La loss sul training è stata di **1.160** mentre sul validation **2.057**.

## Seconda fase Sparse

Nella seconda fase Sparse i modelli sono stati addestrati per 50 epoche. Per il modello della VGGFace, il valore di learning rate ottenuto dalla fase precedente è stato di **1e-30**, mentre per il modello di ResNet50 è stato di **1e-29.**

****

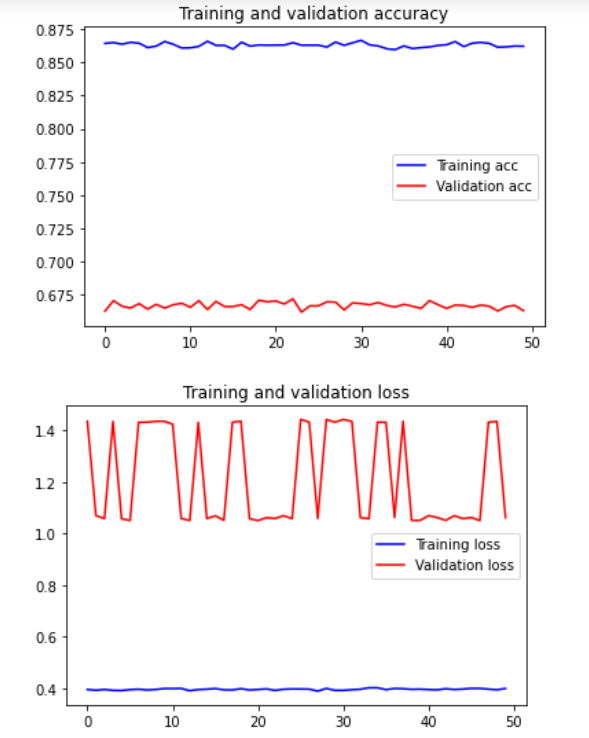
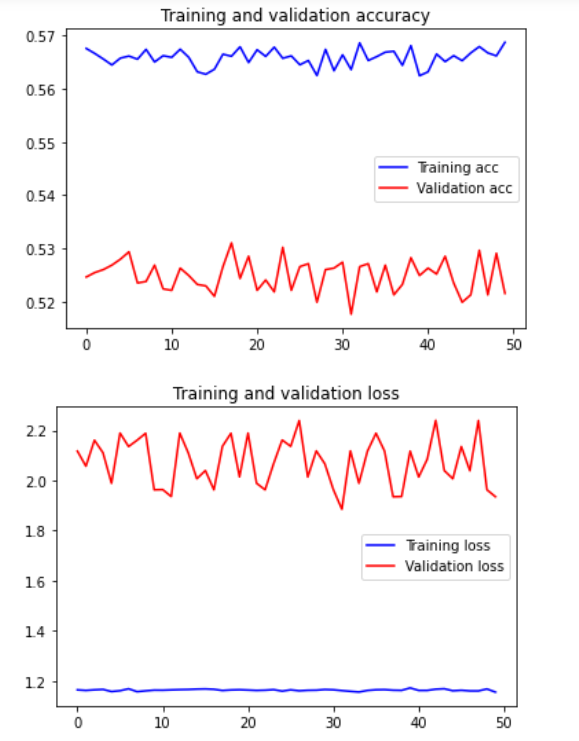
Risultati relativi al modello VGGFace Risultati relativi al modello ResNet50

Nella seconda fase Sparse, per la VGGFace si è ottenuto un valore di accuracy sul training pari a **0.816**, mentre l’accuracy sul validation è stata di **0.648.** La loss sul training è stata di **0.522**, mentre quella sul validation è stata di **0.907**.

Per quanto riguarda il modello ResNet50, il valore di accuracy sul training è di **0.566**, mentre sul validation è di **0.527**. La loss sul training è stata di **1.162** mentre sul validation **1.989.**

## Terza fase Dense

Infine, nella terza fase Dense, i modelli sono stati addestrati per altre 50 epoche, con un totale di 400 epoche. Per il modello della VGGFace il valore di learning rate ottenuto è stato di **1e-35**, mentre per il modello di ResNet50 è stato di **1e-34.**

****

Risultati relativi al modello VGGFace Risultati relativi al modello ResNet50

Nella terza ed ultima fase Dense, per la VGGFace si è ottenuto un valore di accuracy sul training pari a **0.862**, mentre l’accuracy sul validation è stata di **0.663.** La loss sul training è stata di **0.399**, mentre quella sul validation è stata di **1.062**.

Per quanto riguarda il modello ResNet50, il valore di accuracy sul training è di **0.568**, mentre sul validation è di **0.522**. La loss sul training è stata di **1.155** mentre sul validation **1.935.**

## Conclusioni

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **training\_acc** | **training\_loss** | **val\_acc** | **val\_loss** |
| **Dense 1** | 0.861 | 0.395 | 0.6626 | 1.44 |
| **Sparse 1** | 0.813 | 0.527 | 0.653 | 0.926 |
| **Dense 2** | 0.862 | 0.396 | 0.666 | 1.425 |
| **Sparse 2** | 0.816 | 0.522 | 0.648 | 0.907 |
| **Dense 3** | **0.862** | **0.399** | **0.663** | **1.062** |

Tabella riassuntiva modello VGGFace

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **training\_acc** | **training\_loss** | **val\_acc** | **val\_loss** |
| **Dense 1** | 0.569 | 1.165 | 0.524 | 2.16 |
| **Sparse 1** | 0.564 | 1.165 | 0.522 | 2.007 |
| **Dense 2** | 0.566 | 1.160 | 0.523 | 2.057 |
| **Sparse 2** | 0.566 | 1.162 | 0.527 | 1.989 |
| **Dense 3** | **0.568** | **1.155** | **0.522** | **1.935** |

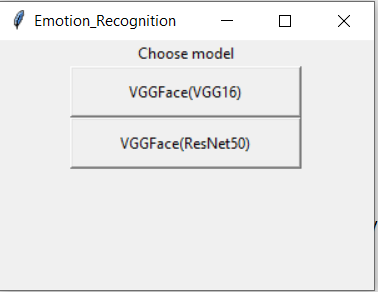
Tabella riassuntiva modello ResNet50

In generale si può osservare come l’applicazione dell’approccio Dense-Sparse-Dense, non abbia apportato miglioramenti nei valori di val\_acc. Infatti nel caso della VGGFace, il valore di val\_acc ottenuto dopo l’ultima fase Dense (**0.663**) è migliore rispetto a quello ottenuto nella prima fase Dense (0.6626), anche se lo scarto tra i due è molto piccolo. Nel caso invece del modello ResNet50, si è ottenuto un lieve peggioramento (**0.522 < 0.524**). Tuttavia ci si deve soffermare anche sui valori di val\_loss ottenuti in entrambi i modelli. Infatti osservando i risultati si evince come l’applicazione della DSD abbia portato ad un netto miglioramento in termini di loss sul validation, riducendo così la probabilità di overfitting dei modelli. Nel caso della VGGFace si è ottenuto un valore di val\_loss, finale, pari a **1.062**, nettamente inferiore rispetto al valore di **1.44** ottenuto dopo la prima fase Dense. Analogo discorso anche per il modello ResNet50, per il quale si è ottenuto un valore di val\_loss finale pari a **1.935**, inferiore rispetto al valore **di 2.16** ottenuto nella prima fase Dense. Per quanto riguarda i valori di acc e loss sul training, non si sono ottenute sostanziali differenze, i valori di tutte le fasi sembrano essere lineari presentando solo delle lievi differenze.

# Riconoscimento live delle emozioni del volto

Una volta aver addestrato i due modelli di VGGFace, è stato creato un codice in Python per effettuare il riconoscimento delle emozioni di una persona in tempo reale. Una volta aver avviato lo script, l’utente può scegliere con quale modello avviare il riconoscimento. Nello script vengono estratti continuamente dei frame dalla webcam, ogni frame viene convertito in un array di pixels i quali vengono passati al modello scelto, inoltre il sistema individua la presenza di un volto, evidenziandolo con un rettangolo. Successivamente il modello, per ogni frame che riceve, effettua la predizione dell’emozione e la mostra all’utente.

**Interfaccia per scelta modello**



**Esempi di riconoscimento live delle emozioni**

