Deep Learning Project- Master Big Data 2022

Giulia Paola Luongo

INTRODUZIONE

Lo scopo dell'analisi è l'addestramento di modelli basati su reti neurali per speech recognition. Ogni parola (30 in questo dataset, in lingua inglese) viene associata a sequenze che ne rappresentano la parola "parlata".

I modelli possono essere utilizzati in ambito di riconoscimento vocale, finalizzato alla comunicazione uomo-macchina: alcuni esempi di applicazione possono essere gli assistenti vocali, la trascrizione simultanea in testo di un discorso parlato o anche la traduzione.

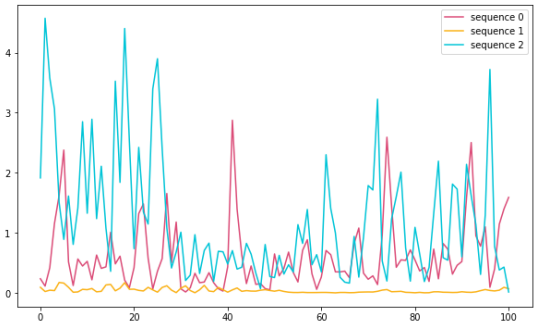
METODO

Caratteristiche Dataset

Il dataset utilizzato contiene dati sequenziali di speech recognition: si compone di 41849 sequenze, ciascuna composta da 101 istanti temporali. Ad ogni istante temporale sono presenti 40 misure.

```
Batch size: 41849
Sequence length: 101
Input size: 40
```

Di seguito una rappresentazione dei dati della prima delle 40 feature, appartenenti alle prime 3 sequenze del dataset.

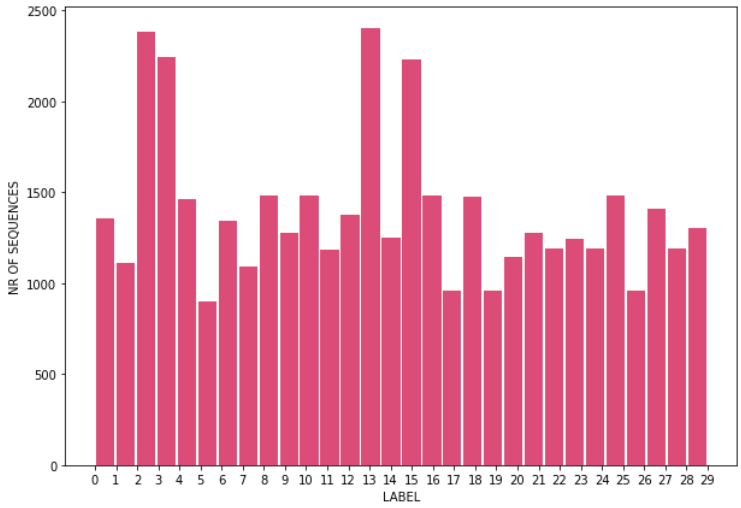


Il dataset si compone di 30 classi, dove ogni classe rappresenta una parola pronunciata. Le parole presenti nel dataset sono: bed, eight, house, off, sheila, two, bird, five, left, one, six, up, cat, four, marvel, on, stop, wow, dog, go, nine, right, three, yes, down, happy, no, seven, tree, zero.

La tabella e il grafico seguenti mostrano la distribuzione dei dati, ovvero il numero di sequenze presenti nel dataset per ognuna delle 30 parole.

LABEL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
NR OF SEQUENCES	1356	1113	2382	2244	1463	902	1346	1092	1485	1276	1485	1187	1378	2400	1253	2228	1485	957	1474	960

La distribuzione dei dati è piuttosto uniforme: la maggior parte delle classi (26) è composta da un numero di sequenze compreso tra 1000 e 1500. Quattro classi presentano invece un numero di osservazioni compreso tra 2000 e 2500.



Per l'analisi effettuata in questo progetto non sono state effettuate operazioni di undersampling e/o oversampling per uniformare il numero di dati per classe. Tuttavia questo potrebbe essere un possibile miglioramento da implementare in futuro.

Suddivisione del Dataset

Allo scopo della presente analisi, è stato effettuato uno split dei dati fra training set e test set. La threshold è stata impostata al 25%, quindi il 75% dei dati sono stati utilizzati come training e il restante come test. Sono stati impostati il parametro "shuffle", che divide i dati in modo casuale e non ordinato, e la stratificazione in base alle classi, in modo da avere lo stesso bilanciamento dei dati per classe anche a seguito della suddivisione. Ciò è utile anche alla luce del fatto che il numero di dati per categoria non sia completamente uniforme, come sopra descritto.

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((31387, 101, 40), (31387,), (10462, 101, 40), (10462,))
```

In seguito è stata effettuata un'ulteriore suddivisione dei dati di training per ottenere un validation set: in questo caso il threshold è stato impostato al 20%, mentre come per lo split precedente, è stato impostato il parametro shuffle e la stratificazione.

```
x_train.shape, y_train.shape, x_valid.shape, y_valid.shape
```

```
((25110, 101, 40), (25110,), (6277, 101, 40), (6277,))
```

Normalizzazione

È stata effettuata una normalizzazione dei dati, utilizzando la classe Normalization del modulo layers di keras. La normalizzazione è stata "adattata" sui dati di training e poi applicata ai dati di train, test e validation.

```
Training data
Mean:  1.4876674e-08
Std:  1.0000006
```

```
Test data
Mean:  -0.005306041
Std:  0.9835531
```

```
Validation data
Mean:  -0.0014269846
Std:  0.9840288
```

Modelli

Le architetture utilizzate in questa analisi sono MLP (Multi-Layer Perceptron) e CNN (Convolutional Neural Network). In entrambi i casi è stata effettuata una prima fase di selezione degli iperparametri ottimali, attraverso la classe ParameterGrid di sklearn.

MLP

Il modello MLP è stato strutturato in questo modo:

1. Inizializzazione del modello
2. Layer di input
3. Reshape
4. Tre layer densi con funzione di attivazione "tanh"
5. Ultimo layer di classificazione, con funzione di attivazione "softmax"

6. Compilazione del modello con ottimizzatore “adam”
7. Fit del modello con 10 epoche

Sono stati testati i seguenti iperparametri, con i seguenti valori:

- Batch Size: 16, 32, 64
- Nr of units (neuroni) primo layer denso: 32, 64, 128
- Nr of units (neuroni) secondo e terzo layer densi: 32, 64, 128

per un totale di 27 combinazioni.

CNN

Per quanto riguarda invece il modello CNN, è stata utilizzata la seguente struttura:

1. Layer di definizione input
2. Primo blocco, con layer Conv1d con 16 filtri e padding, e layer ReLU
3. Secondo blocco, con layer Conv1d con 32 filtri e padding, e layer ReLU
4. Terzo blocco, con layer Conv1d con 64 filtri e padding, e layer ReLU
5. Blocco finale con layer GlobalAveragePooling1D e con layer Denso di output
6. Compilazione del modello con ottimizzatore “adam”
7. Fit del modello

Sono stati testati i seguenti iperparametri, con i seguenti valori:

- Batch Size: 4, 8, 16, 32
- Epoche: 10, 15, 20
- Kernel Size: 2, 4

per un totale di 24 combinazioni.

RISULTATI: Model selection e assessment

MLP

Dalla fase di hyperparameter tuning risulta la seguente combinazione di parametri come quella con loss migliore:

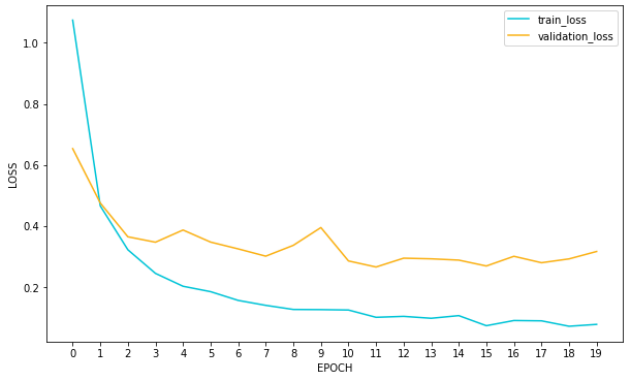
- Batch Size: 16, 32, **64**
- Nr of units (neuroni) primo layer denso: 32, **64**, 128
- Nr of units (neuroni) secondo e terzo layer densi: 32, 64, **128**

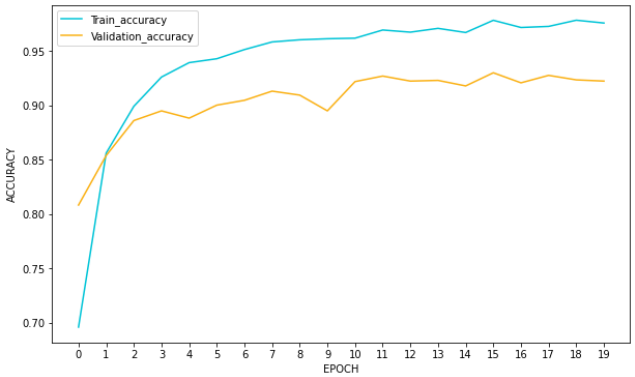
```
BEST LOSS:  0.2996390461921692
BEST CONF:  {'bs': 64, 'n1': 64, 'n2': 128}
```

I dati sono quindi stati utilizzati nella costruzione del modello.

```
Epoch 20/20
393/393 [==============================] - 3s 8ms/step - loss: 0.0783 - accuracy: 0.9757 - val_loss: 0.3169 - val_accuracy: 0.9223
```

Di seguito i plot di loss e accuracy nel corso delle 20 epoche.





Si può notare come le performace sul validation set siano leggermente peggiori rispetto al training set: ciò può significare la presenza di overfitting del modello sui dati di training, cosa che causa una classificazione peggiore dei dati di validation.

Il modello è stato poi applicato ai dati di test, ottenendo il seguente risultato.

```
327/327 [==============================] - 1s 2ms/step - loss: 0.3252 - accuracy: 0.9187
[0.32523807883262634, 0.9186580181121826]
```

Si ottiene un'accuracy del 92%, rispetto a un valore del 97% ottenuto sui dati di training: anche in questo caso si nota un leggero overfitting.

CNN

I parametri risultati come ottimali dalla fase di hyperparameter tuning sono:

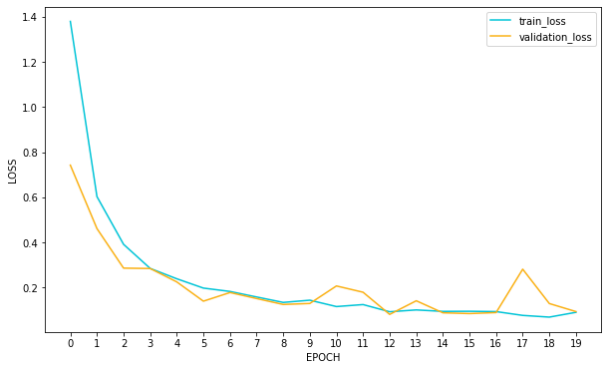
- Batch Size: **4**, 8, 16, 32
- Epoche: 10, 15, **20**
- Kernel Size: 2, **4**

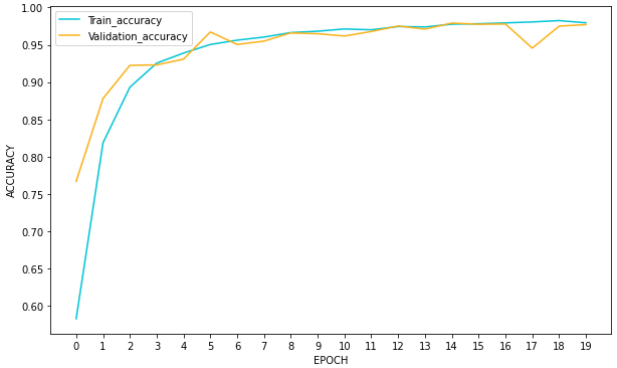
```
BEST LOSS:  0.08250086009502411
BEST CONF:  {'bs': 4, 'ep': 20, 'k': 4}
```

I dati sono quindi stati utilizzati nella costruzione del modello.

```
Epoch 20/20
6278/6278 [==============================] - 52s 8ms/step - loss: 0.0894 - accuracy: 0.9795 - val_loss: 0.0927 - val_accuracy: 0.9771
```

Di seguito i plot di loss e accuracy nel corso delle 20 epoche.





In questo caso le performace sul validation set non presentano grosse differenze rispetto al training set.

Infine il modello è stato applicato ai dati di test, ottenendo un'accuracy del 97,75%.

```
327/327 [==============================] - 2s 7ms/step - loss: 0.1106 - accuracy: 0.9775
[0.11055858433246613, 0.9775377511978149]
```

CONCLUSIONI

Tra i due modelli utilizzati, le performance migliori sono state ottenute con l'architettura CNN: in particolare è stata ottenuta un'accuracy del 97,75%, contro il 91,87% ottenuto tramite modello MLP. Inoltre come già anticipato, il modello MLP presentava un leggero overfitting.

Un possibile miglioramento di questo progetto sarebbe quindi l'effettuazione di ulteriori prove per la riduzione dell'overfitting.

Un altro possibile miglioramento, come riportato all'inizio del report, sarebbe l'uniformazione del numero di dati per ogni classe, effettuando undersampling delle categorie con più dati, e eventuale oversampling delle categorie con meno dati.