Assignment 2

Autoencoders and NN on P-Z letters

Introduzione

Il progetto è stato realizzato su google **colab** in **python**, servendosi delle librerie Keras, **Keras_metrics** (è già presente il comando pip per installarla) e **Pandas**.
I Path dei dtreue file (x_train.obj , y_train.obj e x_test.obj) vanno modificati prima dell'esecuzione: nel caso riportato essi sono presi direttamente dal mio Google Drive e importati tramite pickle.

Analisi del Dataset

```
[ ]  from collections import Counter
     count = Counter(y_train)
     count
```

```
Counter({16: 1295,
         17: 1265,
         18: 1346,
         19: 1329,
         20: 1336,
         21: 1297,
         22: 1269,
         23: 1327,
         24: 1322,
         25: 1321,
         26: 893})
```

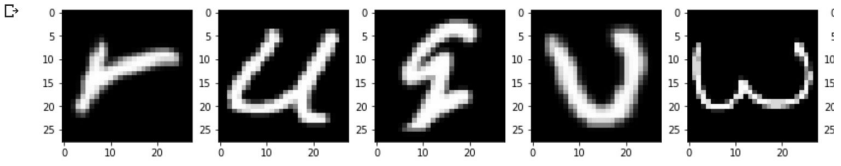
Il dataset presenta

- **14.000** matrici 28x28 come dati di **train**
- **14.000 label** dei dati di **train**
- **8.800** matrici 28x28 come dati di **test** (senza label)

Il DS sembra essere **bilanciato** per quanto riguarda i dati di train.

Le label si riferiscono all'**ordine alfabetico** delle lettere rappresentate. Ho realizzato una funzione che dato un numero ne restituisce la **lettera corrispondente** alfabetica, così da poter confrontare più comodamente i risultati successivamente.

```
[ ]  n = 10
     plt.figure(figsize=(28, 28))
     for i in range(n):
         ax = plt.subplot(2, n, i + 1)
         plt.imshow(x_train[i].reshape(28, 28))
         plt.gray()
```



Reshape dei dati

```
Shape prima del reshape:
(14000, 28, 28) Train Set
(8800, 28, 28)  Test Set

Shape dopo il reshape:
(14000, 784) Train Set
(8800, 784)  Test Set
```

I dataset forniti sono stati **ridimensionati** per essere utilizzati per il learning con keras.
In particolare, le matrici 28x28 sono state trasformate in un singolo array da 728pixel.

Creazione del Validation Set

Il dataset di **train** è stato splittato come segue:

- **85%** traning set
- **15%** validation set

Nella figura a destra è possibile vedere nel dettaglio la divisione tra train e validation set, ovvero:

- **11.900** elementi nel traning set
- **2.100** elementi nel validation set

```
-> Original Train Set Shape:
(14000, 784)
(14000, 784)

---- VALIDATION SPLIT ----

-> Train Set Shape:
(11900, 784)
(11900,)
-> Validation Set Shape:
(2100, 784)
(2100,)
```

Creazione del modello dell'autoencoder

```
# Rappresentazione encoded (codificata) dell'inpunt
encoded = Dense(encoding_dim, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)

# Ricostruzione (lossy) dell'input
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(784, activation='sigmoid')(decoded)
```

L'autoencoder ha **due livell**i per la parte di *encoding* e **due livelli** per la parte di *decoding*.
L'encoder produce immagini da **64pixel** rispetto a quelle da 784pixel iniziali con un notevole risparmio di dati.
La funzione di attivazione è **relu** per tutte i layer tranne nell'ultimo layer del decoder dove invece è utilizzata la funzione di attivazione **sigmoid**.

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()
```

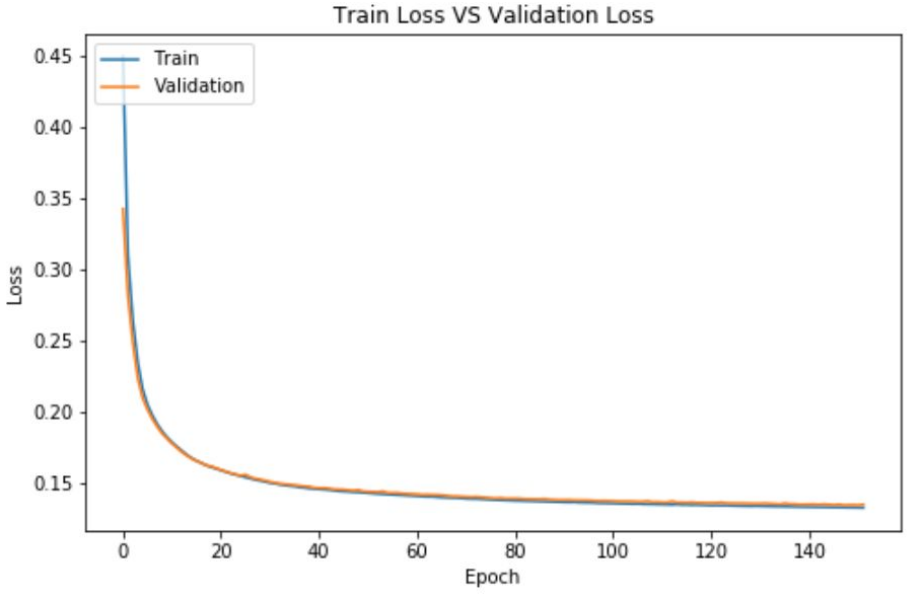
```
Model: "model_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        (None, 784)               0
_________________________________________________________________
dense_24 (Dense)             (None, 64)                50240
_________________________________________________________________
dense_25 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_26 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_27 (Dense)             (None, 784)               50960
=================================================================
Total params: 109,520
Trainable params: 109,520
Non-trainable params: 0
```

Fase di training del modello

```
from keras.callbacks import EarlyStopping, ModelCheckpoint
early_stop = EarlyStopping(monitor='val_loss', patience=4, verbose=1) #Imposto

#note: x_train, x_train :)
history = autoencoder.fit(train_data, train_data,
                epochs=200,
                batch_size=200,
                shuffle=True,
                validation_data=(val_data, val_data), callbacks=[early_stop])
```

La **binary_crossentropy** alla fine della fase di fit è:

- loss: **0.1330**
- val_loss: **0.1352**

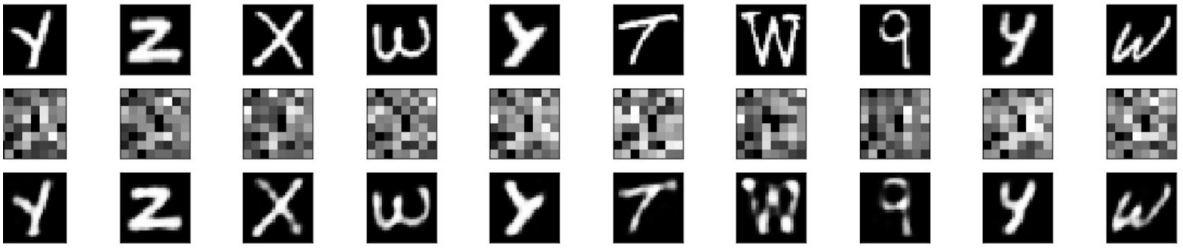


I valori delle loss sembrano essere abbastanza bassi e il loro **decremento** è quasi parallelo. Le epoche impostate come caso limite sono **200** ma interviene sempre l'**early stop** ad anticipare la fine della fase di fit. L'early stop agisce dopo che il valore di validation loss **smette di migliorare per 4 volte di fila**.
Il bach_size impostato a 200 sembra dare i risultati migliori: provando a variarlo i valori delle loss precedenti tendono a peggiorare leggermente.

Encoding/Decoding - Confronto visivo

```
encoded_imgs = encoder.predict(train_data)
decoded_imgs = decoder.predict(encoded_imgs)
```

- **Prima riga**: Immagine originale (28x28=**784pixel**)
- **Seconda riga**: Output dell'encoder (8x8=**64pixel**)
- **Terza riga**: ricostruzione del decoder (28x28=**784pixel**)



Le performance dell'autoencoder sono **soddisfacenti**. Le immagini ricostruite (ultima riga) risultano essere abbastanza **fedeli alle originali**.
Si nota però qualche **difetto** comprensibile in alcune lettere "articolate" come la W in terzultima posizione.

Modello per la predizione del carattere

L'obiettivo ora diventa creare una rete in grado di interpretare il carattere corrispondente a partire da una immagine dello stesso.
L'input per la suddetta rete può essere di due tipi:

- Le immagini **originali** da 728pixel
- Le immagini **dell'encoder**, da 64 pixel

Sicuramente usare le immagini originali mi da un vantaggio in termini di **performance** predittive visto che ho a disposizione più pixel, e quindi più dati, su cui effettuare il learning (essendo l'encoding una tecnica di tipo lossy)

Usare le immagini dell'encoder però mi consente di avere un notevolissimo **risparmio** prestazionale in termini di **computazione** essendo che utilizzo solo 64pixel e non 784pixel. (con pixel si intendono comunque valori numerici da 0-255, lavorando in scala di grigi)

Visto che l'autoencoder precedentemente analizzato fornisce tutto sommato buone performance di ricostruzione, nonostante sia una tecnica **lossy**, ho deciso di usare le immagini più piccole come input del modello seguente, ovvero **quelle fornite dall'encoder** (riga centrale dell'immagine precedente).

In teoria mi aspetto che un encoder ben fatto fornisca immagini con performance **simili da quelle che si avrebbero con quelle originali** (non uguali essendo comunque una tecnica lossy), ma con velocità di elaborazione superiore.

Definizione del modello

```
model = Sequential()
model.add(Dense(128, activation='relu',input_shape=(64,) ))
model.add(Dense(32, activation='relu' ))
model.add(Dense(11))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_7 (Dense)              (None, 11)                363
_________________________________________________________________
activation_1 (Activation)    (None, 11)                0
=================================================================
Total params: 12,811
Trainable params: 12,811
Non-trainable params: 0
_________________________________________________________________
```

Dopo **vari tentativi** ho trovato un modello con buone performance.
I vari tentativi mi hanno consentito di arrivare ad una soluzione composta da 3 layer, con attivazione **relu** e **softmax** sull'ultimo layer da **11 neuroni** (11 come le lettere da predire).
I primi due layer invece hanno 128 e 32 neuroni rispettivamente: diminuendo il numero di neuroni ho notato un leggerissimo peggioramento quindi ho deciso di lasciare questa composizione.

La loss scelta è la **sparse_categorical_crossentropy** poichè le classi da predire sono degli interi non binarizzati (non one-hot encoded).
Infatti, se fossero stati **one-hot encoded,** avrei potuto usare la categorical_crossentropy.
L'ottimizzatore **ADAM** ha mostrato ottime performance, quindi la scelta è ricaduta su di esso.

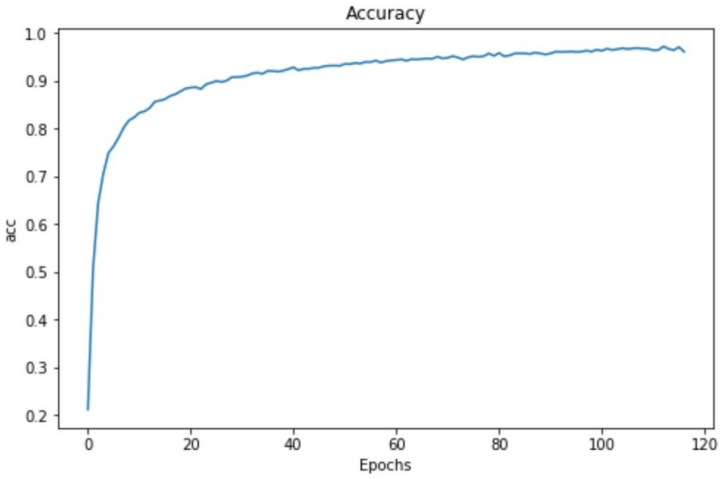
Learning del Modello

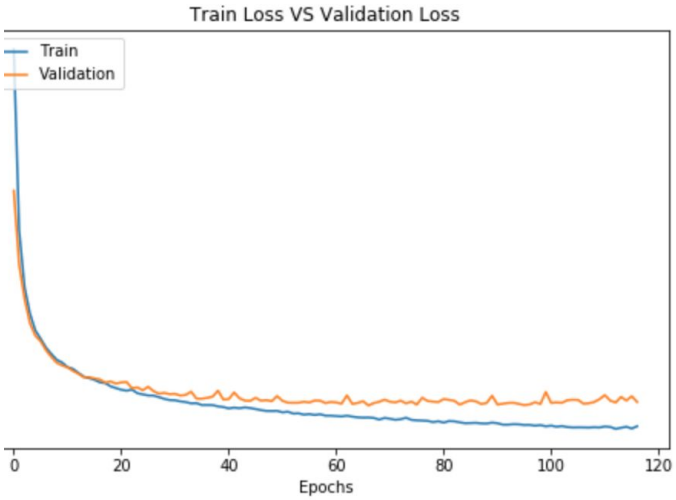
```
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1) #Imposto che le epoche si fermi

history_dieci = model.fit(encoded_imgs, train_target, validation_data=(encoded_imgs_val, val_target),
                          epochs=200, batch_size=160, callbacks=[early_stop])
```

Le performance ottenute dal learning, mantenendo la stessa divisione train e validation di cui sopra, sono i seguenti:

- **loss**: 0.0936
- **acc**: 0.9708
- **val_loss**: 0.3316
- **val_acc**: *0.9243*

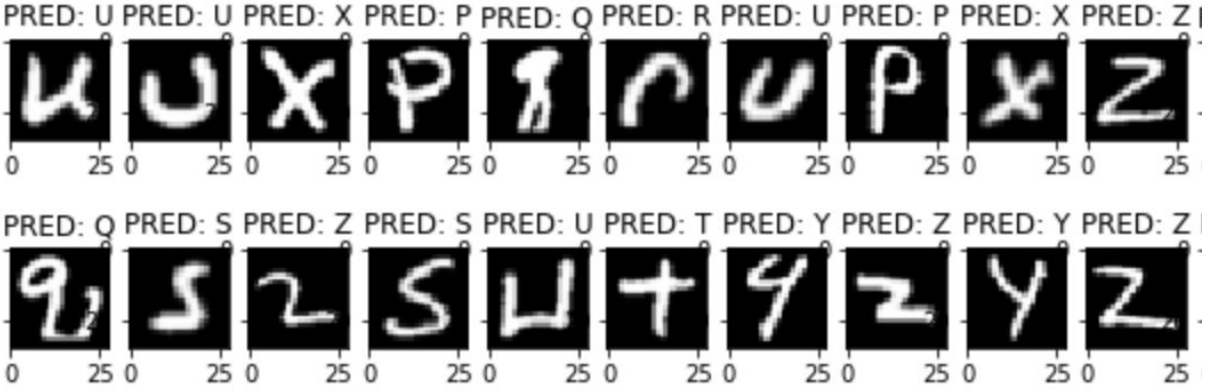
*L'accuracy ottenuta sui dati di validazione è quindi del **92%***

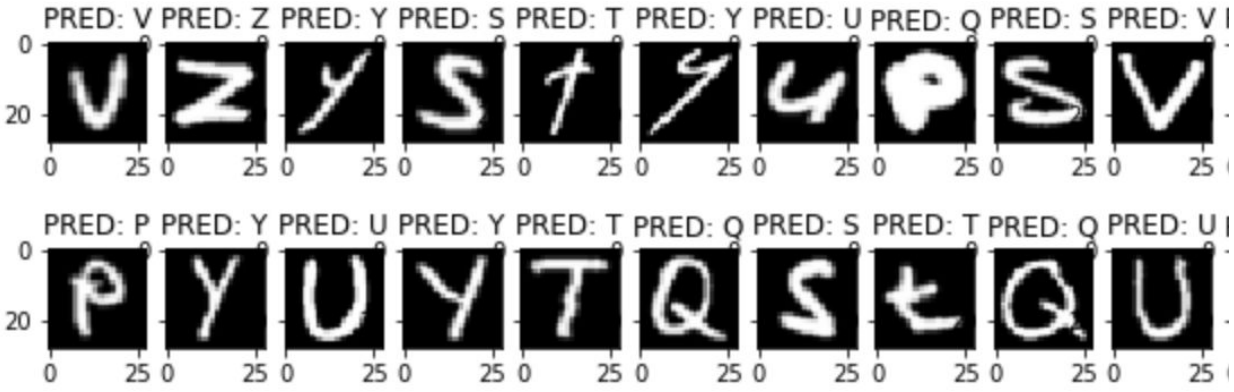




Valutazione delle Performance sul test set

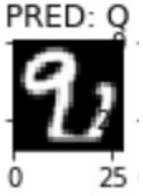
Non avendo a disposizione le label target del test set la valutazione dello stesso può essere fatta su un campione "manualmente" andando a visualizzare cosa ha predetto il modello (PRED: "x") relativamente all'immagine sottostante.





Da un'analisi visiva sul campione considerato il modello sembra avere **ottime performance** su campioni che non ha mai visto anche quando la lettera in questione è scritta in modo piuttosto strano:

Il modello classifica correttamente questa immagine di una Q "strana"



Uno dei campioni che mi ha dato problemi prima di migliorare il modello in termini di layer e numero di neuroni è quello a sinistra, che veniva spesso considerato una "**X**" (come comprensibile visto che lo sembra) al posto che una "**T**".
A seguito di un affinamento dei layer dell'encoder e della rete neurale essa viene considerata, come è giusto che sia, una "T".