Carta Costantino 808417 – Advanced Machine Learning

REPORT – ASSIGNMENT 2

Obiettivi

Gli obiettivi di questo assignment erano i seguenti:

1. Risoluzione del problema di classificazione supervisionata con una rete neurale tradizionale;
2. Creazione di un auto - encoder e visualizzazione e comparazione dell'input originale e dell'output dell'auto – encoder;
3. (opzionale) Uso e valutazione della rappresentazione codificata dell'input generata dalla parte di encoding dell'autoencoder finalizzato all problema di classificazione supervisionata.
4. (opzionale) Generazione di un file .txt con i risultati predetti dal modello di classificazione supervisionata

Dataset

Sono stati forniti tre oggetti di tipo pickle:

1. Dataset di train (14000 matrici 28x28)
2. Label del dataset di train (14000)
3. Dataset di test (8000 matrici 28x28 unlabelled)

Data processing

Sono state eseguite due operazioni principali sui dati:

1. E' stato eseguito il reshape dei dataset forniti con lo scopo di rendere i dati utilizzati usabili per la creazione del modelli di rete neurale.
   In particolare, le matrici 28x28 sono state trasformate in un singolo array da 728 elementi, ciascun elemento rappresentante un pixel dell'immagine.

```
training_set = training_set.reshape((len(training_set), np.prod(training_set.shape[1:])))
validation_set = validation_set.reshape((len(validation_set), np.prod(validation_set.shape[1:])))
test_set = test_set.reshape((len(test_set), np.prod(test_set.shape[1:])))
```

2. I valori x_i delle matrici inizialmente definiti come 0 < x < 255 per ogni x_i sono stati portati a valori tra 0 ed 1.

```
training_set = training_set.astype('float32') / 255.
test_set = test_set.astype('float32') / 255.
validation_set = validation_set.astype('float32') / 255.
```

3. E' stato effettuato il one-hot-encode sulle labels

```
training_labels = to_categorical(training_labels, 11)
validation_labels = to_categorical(validation_labels, 11)
```

Divisione dataset di training

Il dataset di training è stato a sua volta suddiviso in:

- Training -> 80%
- Validation -> 20%

▾ Split training set into train and validation

train = 80% & ***validation*** = 20%

```
[32] training_set, validation_set, training_labels, validation_labels = train_test_split(X_set, Y_train, test_size=0.2, shuffle = 'True', random_state = 33)
```

Creazione autoencoder

L'autoencoder è compost da 8 layer:

- 4 layer di decoding; l'input di 784 pixel viene rappresentato tramite encoding in una immagine a 32 pixel
- 4 layer di decodifica

Per la struttura ho pensato ad una sorta di simmetria tra codifica e decodifica;

Ogni layer ha come funzione di attivazione la '**relu**' a parte l'ultimo in cui è stata usata '**sigmoid**' e come **loss** binary_crossentropy.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_1 (Dense)	(None, 512)	401920
dense_2 (Dense)	(None, 128)	65664
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 64)	2112
dense_6 (Dense)	(None, 128)	8320
dense_7 (Dense)	(None, 512)	66048
dense_8 (Dense)	(None, 784)	402192

Total params: 956,592
Trainable params: 956,592
Non-trainable params: 0

La relu viene utilizzata, per sua natura, solamente nei layer nascosti della rete; E' stata scelta la relu come funzione di attivazione dei layer interni a causa delle sue ottime performance in termini di convergenza.

```
input_img = Input(shape=(784,))
encoded = Dense(512, activation = 'relu')(input_img)
encoded = Dense(128, activation = 'relu')(encoded)
encoded = Dense(64, activation = 'relu')(encoded)
encoded = Dense(32, activation = 'relu')(encoded)

decoded = Dense(64, activation = 'relu')(encoded)
decoded = Dense(128, activation = 'relu')(decoded)
decoded = Dense(512, activation = 'relu')(decoded)
decoded = Dense(784, activation = 'sigmoid')(decoded)
```

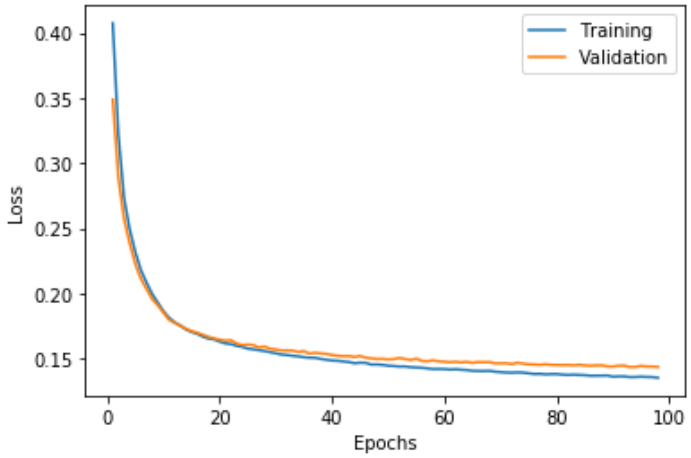
Dopo una serie di esperimenti, visibili nel codice, è stata scelta questa come miglior struttura per l'autoeconder.

Training dell' autoencoder

```
callbacks = [EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)]
net_history = autoencoder.fit(training_set, training_set,
                epochs=100,
                batch_size=256,
                callbacks = callbacks,
                shuffle=True,
                validation_data=(validation_set, validation_set))
```

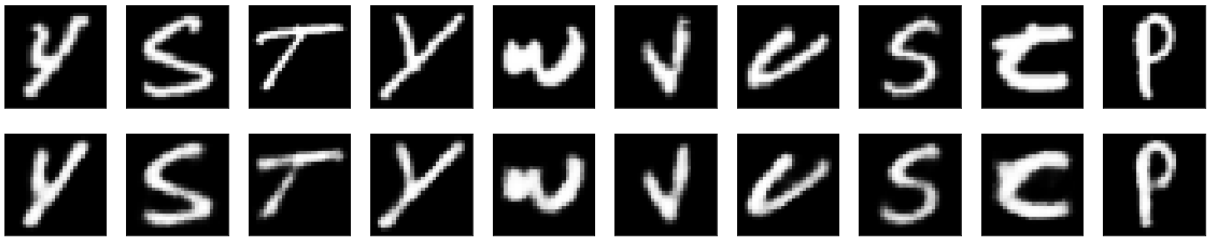
E' stato utilizzato l'early stopping per evitare tendenze all'overfitting durante la fase di training, nel caso per cinque epoche consecutive il valore di loss non migliora la fit viene fermata e vengono restaurati i pesi prima che ciò accadesse.

Performance dell' autoencoder




```
loss: 0.1353 - val_loss: 0.1436
```

Confronto input originale e output dell' autoencoder



Sulla prima riga sono visibili i valori originali di input (10 campioni), nella seconda riga sono visibili gli output ricostruti dalla fase di decoding dopo la fase di encoding.
Come si può notare il modello di autoencoder si comporta egregiamente .

Creazione rete neurale tradizionale

E' stata creata una rete neurale fully connected con lo scopo della classificazione supervisionata sullo stesso training set.

Struttura della rete

```
model = Sequential()
model.add(Dense(784, input_shape=(training_set.shape[1],), activation = 'relu'))
model.add(Dropout(rate = 0.4))

model.add(Dense(512, activation='relu'))
model.add(Dropout(rate = 0.4))

model.add(Dense(256, activation='relu'))
model.add(Dropout(rate = 0.4))

model.add(Dense(128, activation = 'relu'))
model.add(Dropout(rate = 0.3))

model.add(Dense(64, activation = 'relu'))
model.add(Dropout(rate = 0.2))

model.add(Dense(32, activation = 'relu'))
model.add(Dropout(rate = 0.2))

model.add(Dense(11, activation = 'softmax'))

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

La rete è composta da 7 layer ad ognuno dei quali, a parte l'ultimo, è stata applicata una dropout con valore di probabilità di spegnimento dei neuroni che varia in base anche alla dimensione del layer stesso. In questo modo 'aiutiamo' la rete ad apprendere in maniera più ardua e di conseguenza a generalizzare meglio su dati che non ha mai visto.
Funzione di attivazione relu per ogni layer nascosto e softmax per quello di output, in questo modo andiamo a generare delle probabilità per ogni classe.

Performance della rete neurale

Per valutare le performance del modello è stata effettuata una 10 fold cross validation sul dataset di training ed infine il modello migliore è stato testato sul validation set creato inizialmente.

10 fold cross validation

```
for idx, (train, test) in enumerate(kfold.split(training_set, training_labels)):
  print('FOLD: ', idx + 1)
  # create model

  model = Sequential()
  model.add(Dense(784, input_shape=(training_set.shape[1],), activation = 'relu'))
  model.add(Dropout(rate = 0.4))
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(rate = 0.4))
  model.add(Dense(256, activation='relu'))
  model.add(Dropout(rate = 0.4))
  model.add(Dense(128, activation = 'relu'))
  model.add(Dropout(rate = 0.3))
  model.add(Dense(64, activation = 'relu'))
  model.add(Dropout(rate = 0.2))
  model.add(Dense(32, activation = 'relu'))
  model.add(Dropout(rate = 0.2))
  model.add(Dense(11, activation = 'softmax'))
  model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics=['accuracy'])
  callbacks = [EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)]
  # Fit the model
  history = model.fit(
    training_set[train], training_labels[train],
    epochs=200,
    validation_data=(training_set[test], training_labels[test]),
    callbacks = callbacks,
    batch_size=128,
    shuffle=True
  )
  # evaluate the model
  score = model.evaluate(training_set[test], training_labels[test], verbose=1)

  for i in range(2):
    accuracy_results[idx][i] = score[i]
```

Le metriche per ogni modello (accuracy e loss) sono state salvate in una matrice 10x2 in modo tale da verificare poi media e deviazione standard.

Risultati

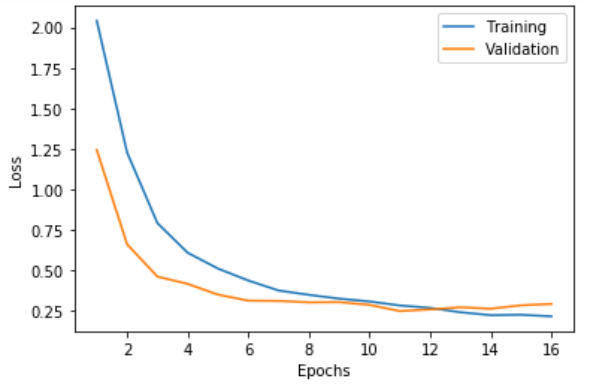
```
array([[0.2462879 , 0.9375    ],
       [0.24739142, 0.94196429],
       [0.30914518, 0.91607143],
       [0.25601345, 0.93839286],
       [0.22168999, 0.94196429],
       [0.23206918, 0.94196429],
       [0.30461972, 0.91964286],
       [0.30884612, 0.91875   ],
       [0.28395978, 0.93214286],
       [0.23302741, 0.94017857]])
```

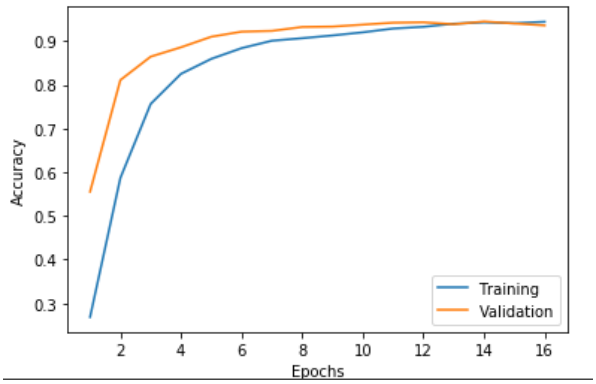
Il modello migliore, in termini di accuracy, e minore loss risulta essere quello trainato alla quinta iterazione dell'algoritmo di cross validation.

Loss = 0.22

Accuracy = 0.94

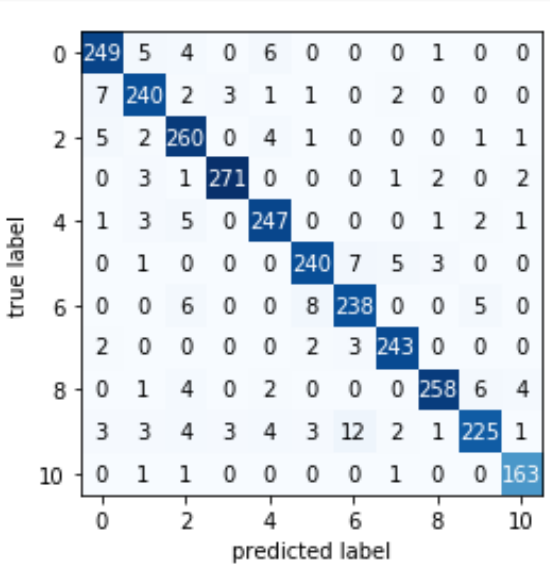
Plot di accuracy e loss del modello migliore





I modello è stato successivamente utilizzato per predirre i valori del validation_set creato inizialmente.

	precision	recall	f1-score	support
P	0.94	0.95	0.94	262
Q	0.93	0.94	0.93	253
R	0.95	0.92	0.93	283
S	0.97	0.98	0.97	276
T	0.95	0.95	0.95	261
U	0.92	0.95	0.93	249
V	0.91	0.93	0.92	250
W	0.96	0.97	0.96	248
X	0.93	0.98	0.95	261
Y	0.85	0.95	0.90	234
Z	0.98	0.96	0.97	168
micro avg	0.93	0.95	0.94	2745
macro avg	0.93	0.95	0.94	2745
weighted avg	0.93	0.95	0.94	2745
samples avg	0.93	0.93	0.93	2745



Utilizzo dell' encoder per il task di classificazione supervisionata

In questa fase è stato presa la parte di encoding dell'autoencoder ed è stata utilizzata come primo layer di una nuova rete neurale.
In questo modo è stata trainata una nuova rete che effettua classificazione sull'input codificato.
I pesi dell'encoder sono stati bloccati, in questo modo rimangono quelli che sono stati trovati quando è stato creato l'autoencoder visto in precedenza.

Creazione del modello

```
model = Sequential()
# First Layer is simply the encoder we trained above
model.add(encoder)
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(11, activation = 'softmax'))

# The weights of the encoder are freezed
model.layers[0].trainable = False

model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics=['accuracy'])

callbacks = [EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)]
# Fit the model
history = model.fit(
  training_set[train], training_labels[train],
  epochs=200,
  validation_data=(training_set[test], training_labels[test]),
  callbacks = callbacks,
  batch_size=128,
  shuffle=True
)
```

La dropout, come in precedenza, è stata utilizzata per permettere al modello di riuscire a generalizzare meglio su dati mai visti.

La rete è composta da un primo layer, che in realtà è **l'encoder** (visibile in figura).

All'encoder sono stati poi collegati tre layer rispettivamente da 32, 16 e 11 (outputt)

```
encoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
=================================================================
Total params: 477,920
Trainable params: 477,920
Non-trainable params: 0
```

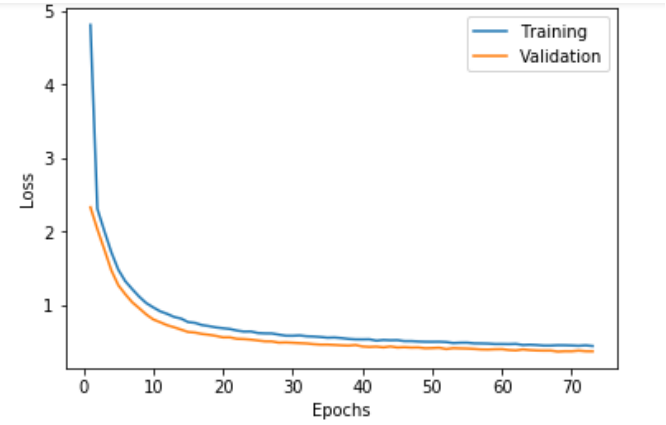
Performance

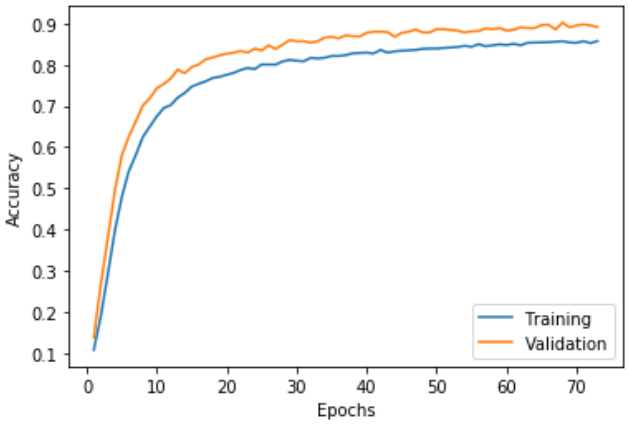
Per valutare le performance è stata utilizzata anche qui la 10 fold cross validation sul **training** set.
Vengono riportati in figura i risultati per ogni fold, rispettivamente **loss** e **accuracy**; (come in precedenza)

```
array([[0.32719368, 0.90178571],
       [0.43413232, 0.87321429],
       [0.36589255, 0.88660714],
       [0.36476302, 0.90267857],
       [0.32483536, 0.90267857],
       [0.33630513, 0.90267857],
       [0.32671202, 0.89910714],
       [0.33162541, 0.88839286],
       [0.4032244 , 0.87857143],
       [0.39708959, 0.87678571]])
```

Come in precedenza il modello migliore in termini di accuracy è stato utilizzato per predirre sul validation set.
Sì può comunque vedere che il modello si comporta in maniera molto simile in termini di accuracy tra i vari fold del training set.
Il metodo è stato uguale a quello visto sopra; il modello viene ricreato ogni volta.
best_acc = 0.90 **best_loss** = 0.32

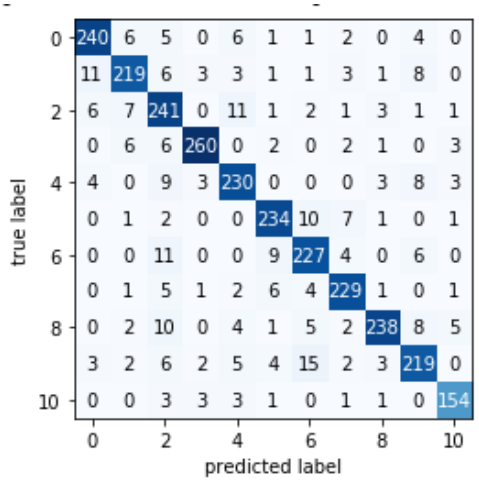
Performance modello migliore sul validation set





I risultati sono abbastanza buoni, nella seconda immagine (accuracy) come ci si poteva aspettare dall'utilizzo della dropout le performance nel validation sono più alte in quanto durante la fase di training alcuni neuroni vengono spenti con p = 0.1 mentre nella fase di validation sono tutti accesi

	precision	recall	f1-score	support
P	0.90	0.93	0.91	256
Q	0.82	0.92	0.87	230
R	0.84	0.85	0.84	270
S	0.91	0.97	0.94	263
T	0.85	0.91	0.88	241
U	0.88	0.93	0.91	242
V	0.84	0.89	0.87	242
W	0.88	0.94	0.91	236
X	0.83	0.97	0.89	237
Y	0.80	0.89	0.84	235
Z	0.92	0.97	0.95	157
micro avg	0.86	0.92	0.89	2609
macro avg	0.86	0.92	0.89	2609
weighted avg	0.86	0.92	0.89	2609
samples avg	0.86	0.86	0.86	2609



n.b = per effettuare la classificazione sul test set è stato utilizzato il modello di rete neurale tradizionale migliore risultante dalla 10 fold cross validation.