Relazione AI

Leonardo Petrilli

Gennaio 2022

1 Introduzione

In questo elaborato viene utilizzata una implementazione di un classificatore **multi-layered perceptron** dalla libreria *scikit-learn*(Python) per applicarlo al problema di riconoscimento di immagini sul dataset *CIFAR 10*.

2 Caratteristiche Dataset

Come già menzionato il dataset utilizzato è CIFAR 10, composto da un totale di 60.000 immagini 32x32 a colori di cui 50.000 predisposte al training della rete neurale e 10.000 per il testing. Ciascuna immagine appartiene a una delle 10 classi in cui è partizionato il dataset e ogni classe contiene 6.000 immagini. Nelle prime righe di codice (16-27) i dati, divisi in *batches*, vengono salvati all'interno di alcune varibili e poi viene fatta una *standardizzazione* con il metodo **scale()**. Infine i dati sono passati in ingresso alle funzioni `fit()` e `predict()` che operano rispettivamente il training e il testing della rete.

3 Struttura della rete neurale

Il modello di rete neurale utilizzato è il **multilayer perceptron** con tre *hidden layers*: il primo layer composto da 2058 neuroni ($\frac{2}{3}$ degli *input neurons* + numero degli *output neurons*), il secondo da 1029 e il terzo da 515. Invece per quanto riguarda l'*input layer* e l'*output layer* essi sono composti rispettivamente da 3072 (1024 x 3, secondo il modello RGB per immagini 32x32 a colori) e 10 neuroni. Un altro iperparametro della rete è il `batch_size` ossia la dimensione dei *minibatches*, che in questo caso è stato impostato al valore 512 dopo alcuni tentativi.
Come già evidenziato in precedenza l'implementazione del classificatore utilizzata è quella della libreria scikit-learn che di default ha le seguenti caratteristiche:

- **Funzione di attivazione (hidden neurons)** → ReLU
- **Solver** → Adam

- **Learning rate** = 0,001
- **Funzione Loss** → Cross-Entropy
- **Funzione di risposta** → Softmax (per classificazione multiclasse)

4 Risultati sperimentali

I risultati raccolti dopo il testing sono:

- **Accuratezza** → $\sim$ 40%
- **Numero di errori** → $\sim$ 6.000
- **Numero di iterazioni** → $\sim$ 150
- **Matrice di confusione**

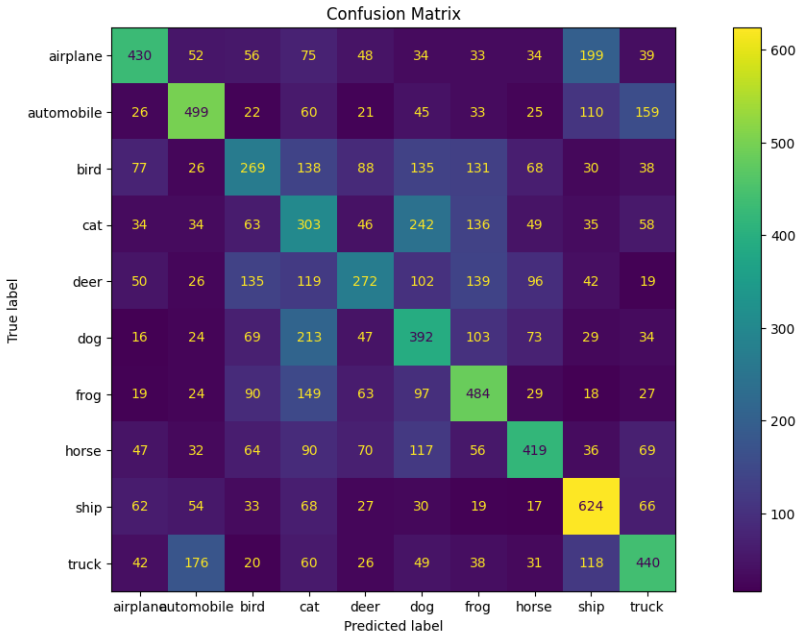

Figure 1: Esempio di matrice di confusione

- **Funzione Loss**

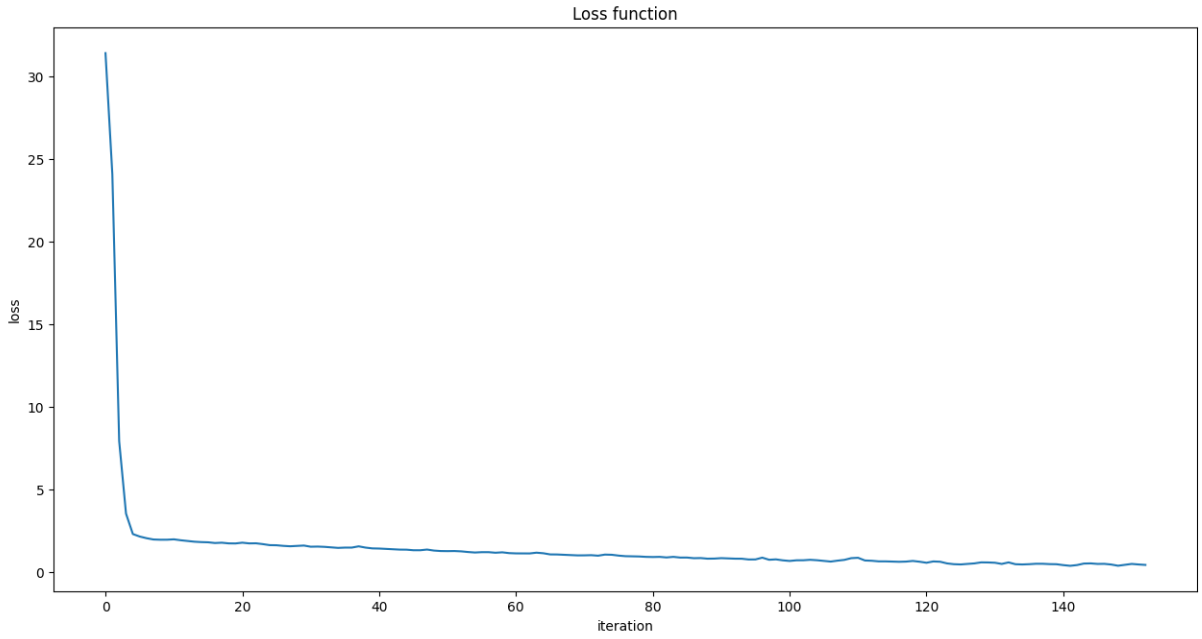

Figure 2: Valori della loss in funzione del numero di iterazioni

Accuratezza e matrice di confusione vengono calcolati rispettivamente tramite i metodi `accuracy_score()` e `confusion_matrix()` già implementati in scikit-learn, la funzione loss è memorizzata all'interno dell'attributo `loss_curve_` della classe *MLPClassifier* mentre il numero di errori è calcolato nel seguente modo:

```
numErrors = 10000 - sum(np.diag(confusionMatrix))
```

Figure 3: 10.000 è il numero di *samples* per il testing