# ****Relazione sul Progetto Fashion MNIST con TensorFlow****

## ****Introduzione****

Questa relazione descrive il lavoro svolto per sviluppare e confrontare due modelli di classificazione di immagini utilizzando il dataset **Fashion MNIST**, disponibile tramite l’interfaccia tf.keras.datasets (https://www.tensorflow.org/datasets/catalog/fashion\_mnist?hl=it).

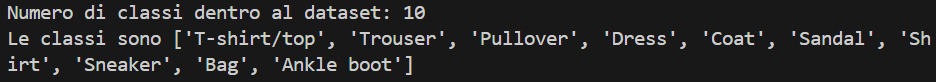
Il progetto consiste in tre script Python:

1. crea\_modello.py: Prima versione del modello, con analisi esplorativa dei dati, addestramento e valutazione.
2. crea\_modello\_2.py: Seconda versione migliorata, con architettura più complessa e prestazioni superiori.
3. usa\_modello.py: Script per caricare il modello salvato ed eseguire inferenza sul dataset di test.

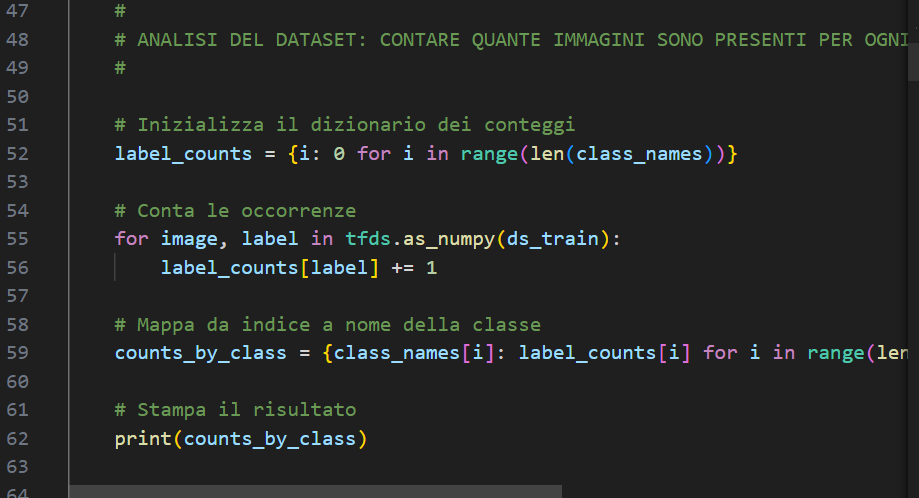
### ****Caricamento e Preprocessing****

* Il dataset è stato caricato utilizzando tfds.load('fashion\_mnist'), suddividendolo in:
  + **Training set** (60.000 immagini)
  + **Validation set** (5.000 immagini, estratto dal test set originale)
  + **Test set** (5.000 immagini rimanenti)
* Le immagini sono in scala di grigi (28×28 pixel) e rappresentano 10 categorie di capi d'abbigliamento:

Printate poi i nomi delle classi e il loro numero.

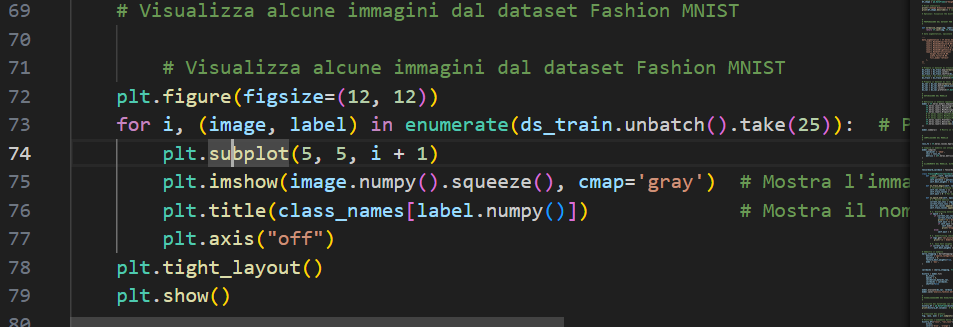
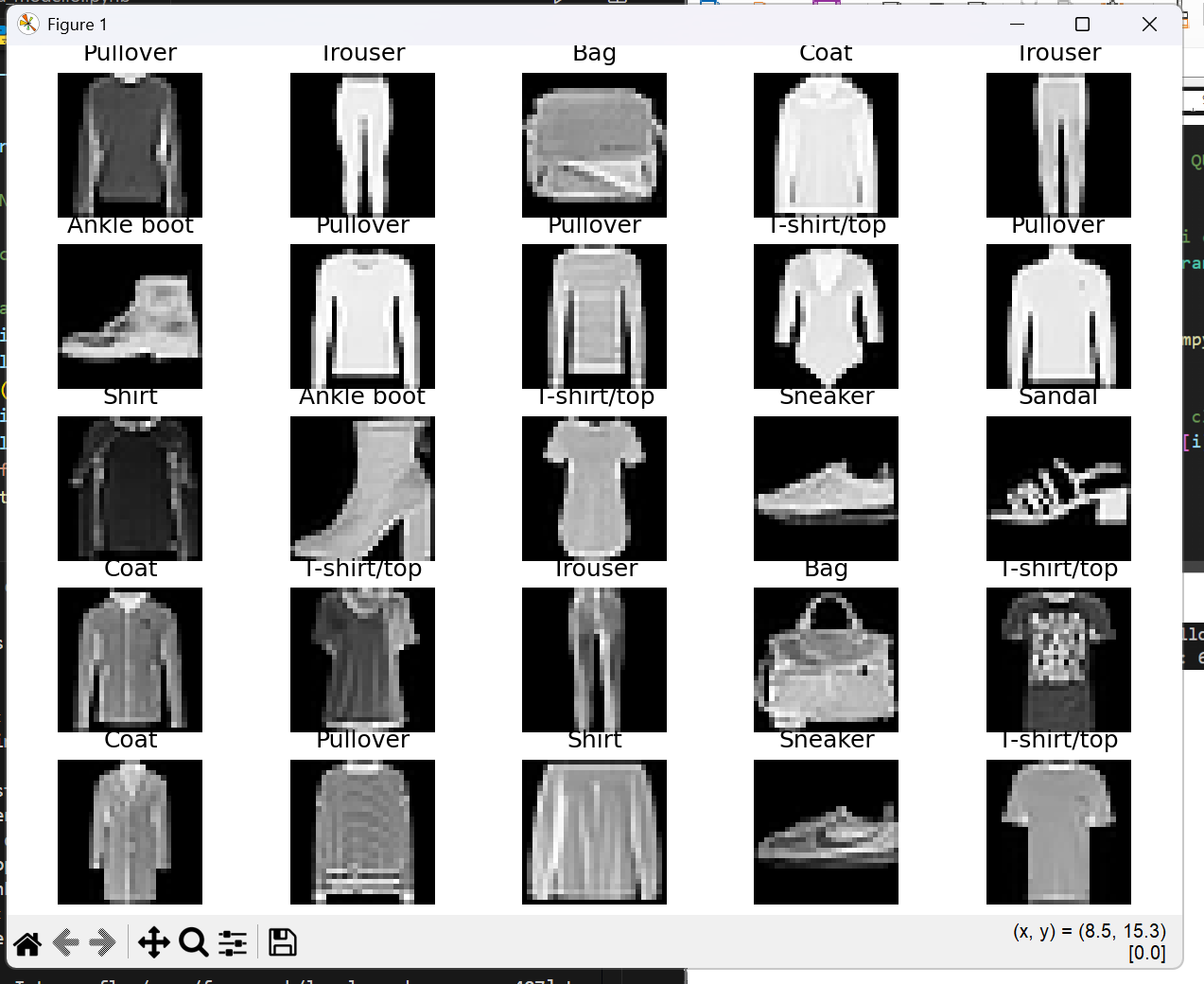


Si passa successivamente all’analisi del dataset, si contano quante immagini sono presenti all’interno di ciascuna classe,

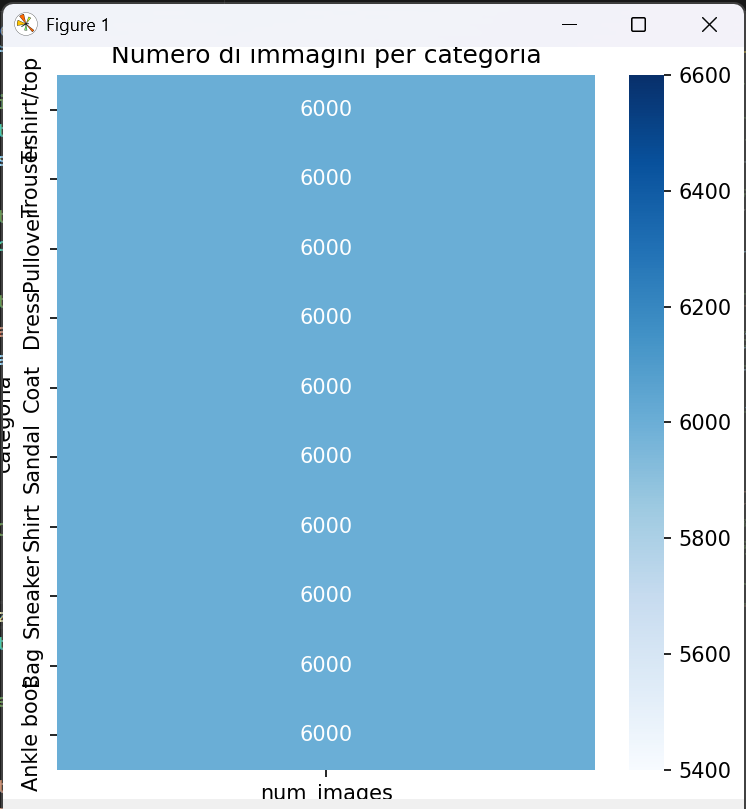
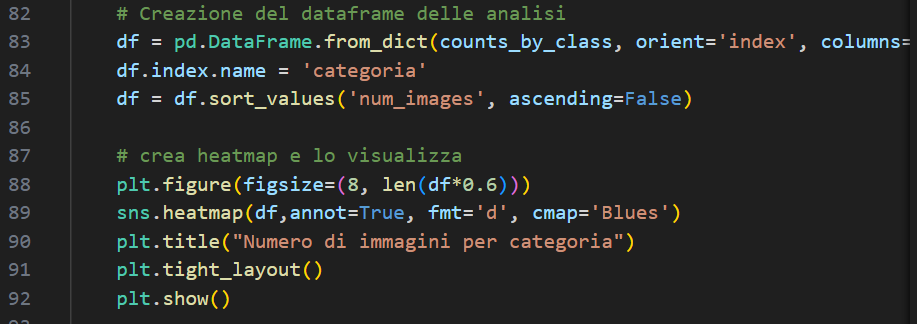




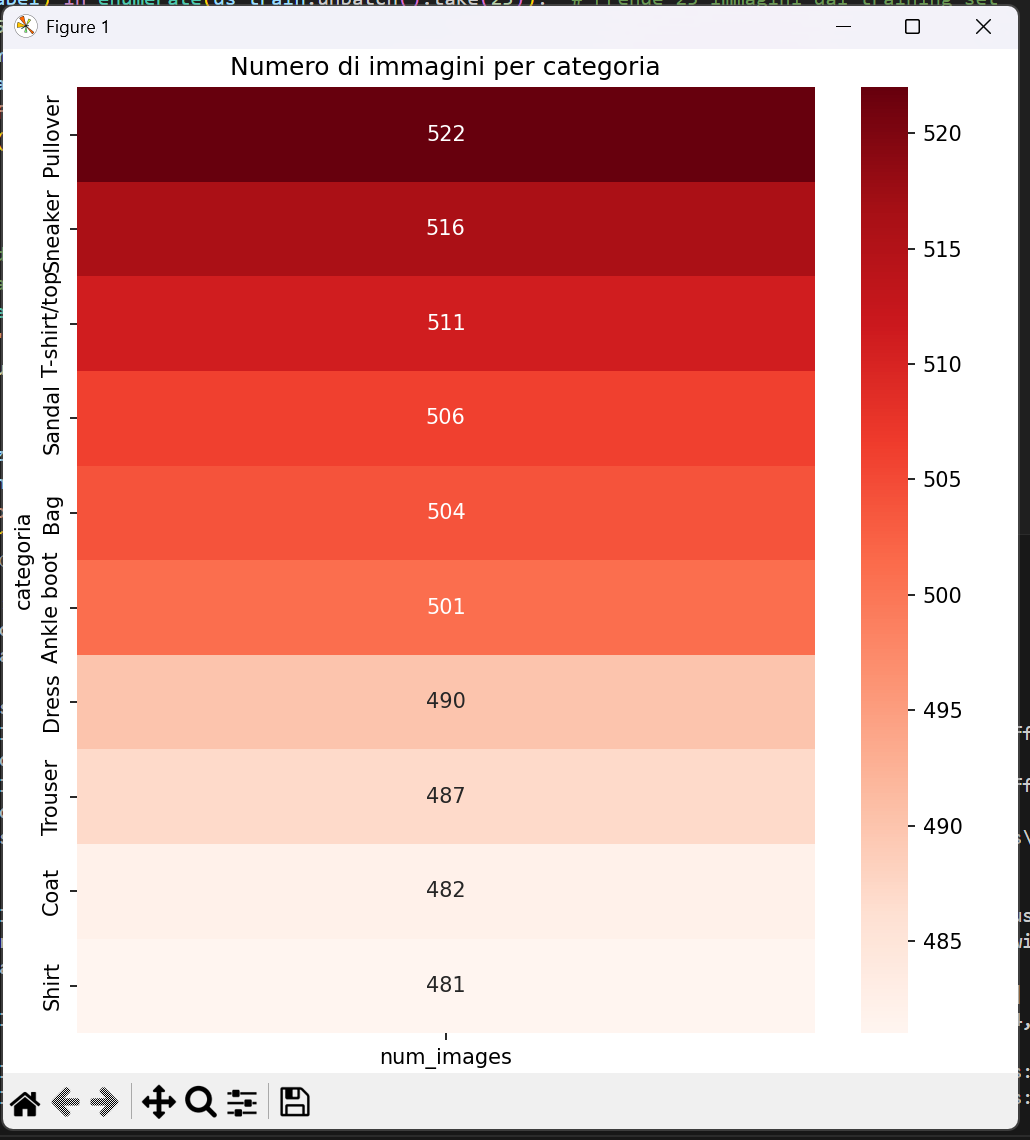
e si plotta un sample del dataset



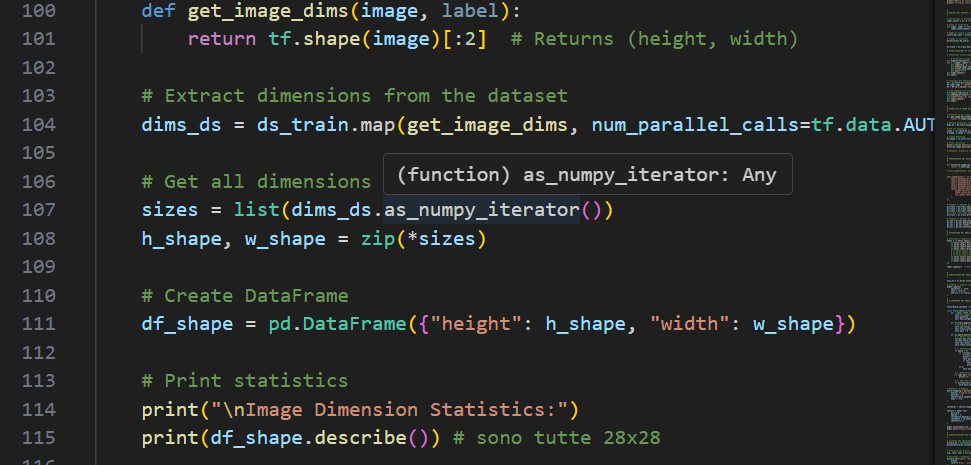
dal dizionario creato precedentemente per il conteggio delle immagini nelle classi, si crea un dataframe e una visualizzazione



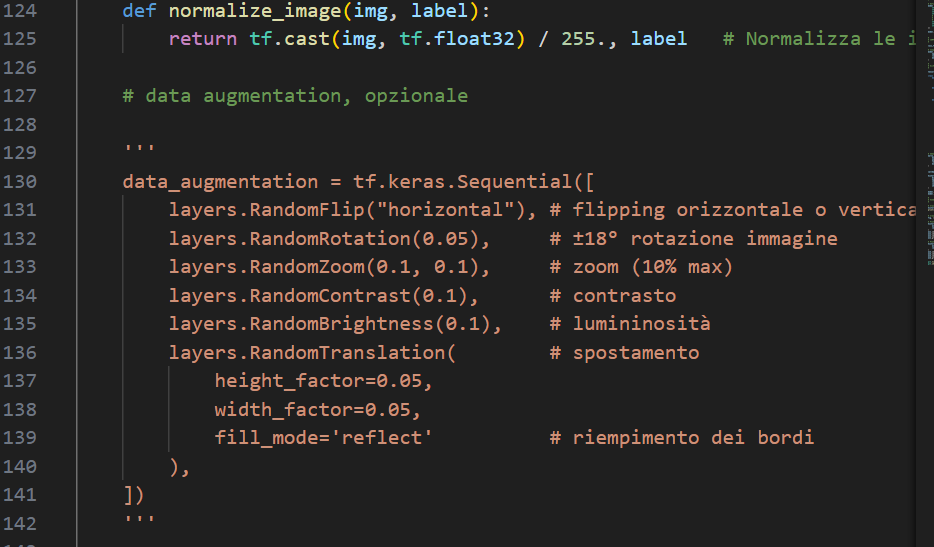
il dataset del train risulta perfettamente bilanciato, ogni classe ha lo stesso numero di immagini al suo interno.

L’operazione è eseguita anche sul dataset di test, e qui le classi non risultano invece perfettamente bilanciate

successivamente si studia la shape delle immagini

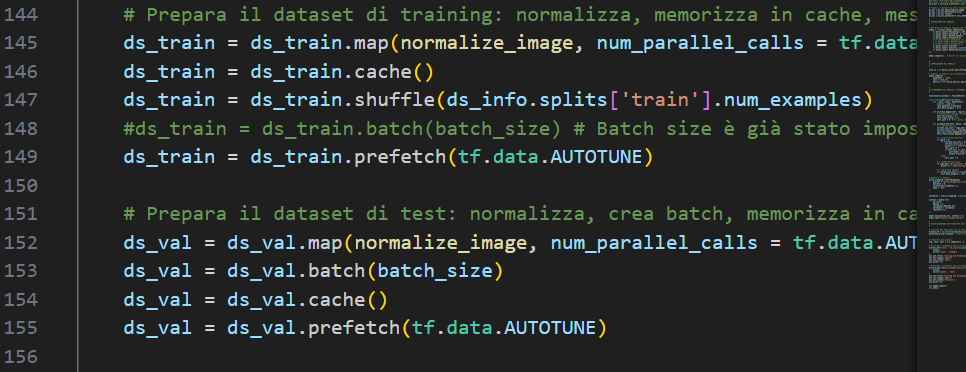
in questo caso sono tutte 28x28, ma questa parte sarebbe utile in caso di dataset più eterogenei per poter studiare delle shape di input diverse nel modello.

Adesso si passa alla preparazione dei dati per il training

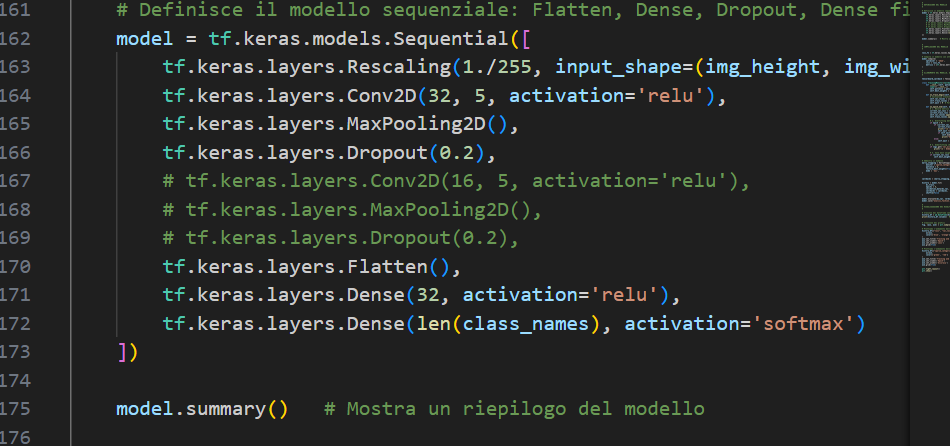


la funziona normalize\_image, normalizza l’immagine per facilitarne la computazione. C’è anche una parte di data augmentation che in alcuni dataset permette una migliore qualità del training, specialmente in dataset più piccoli, andando a modificare alcune caratteristiche delle immagini di training, come contrasto, luminosità, forma, rotazione ecc. questa parte se implementata deve essere aggiunta come layer nella formazione del modello.

Ancora, vengono ottimizzati i dati in input per velocizzare la computazione



successivamente vengono definiti gli strati del modello:



- il primo strato è sempre quello di input, dove vengono passate le immagini della shape selezionata e coerente con quella del training.

- Il secondo strato è uno convoluzionale, 32 filtri con un pattern 5x5 che fanno uno scan dell’input - layer e ne estraggono le caratteristiche principali e caratteristiche.

- Il terzo strato è un pooling, che riduce di dimensionalità le immagini e ne estrae (da un filtro 2x2 standard) i valori max, per propagare le caratteristiche più significative.

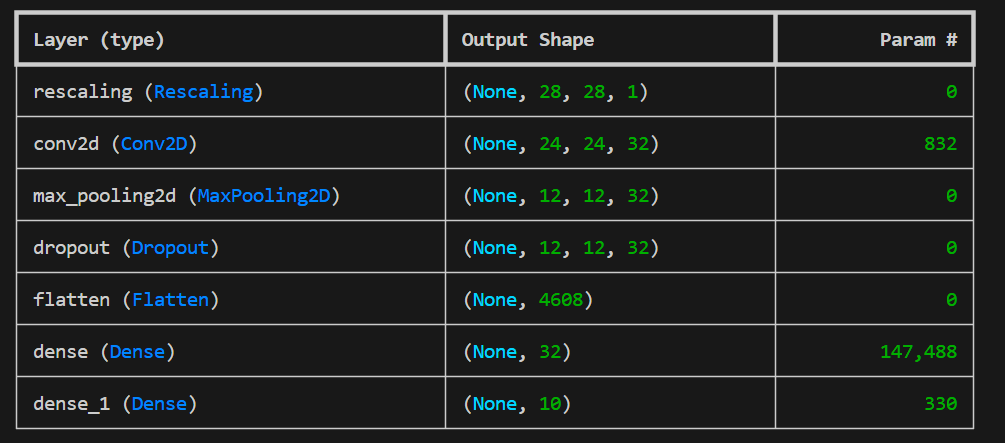
- Il quarto strato è di dropout per evitare che si formi overfitting, si spengono alcuni neuroni per non far imparare a memoria il modello (troppa esercitazione sul dataset del train)

- il quinto strato appiattisce l’immagine matrice in un vettore

- il sesto strato dense (dove si formano la maggior parte dei collegamenti) mette in relazione 32 neuroni con ogni strato flattened che arriva dagli strati successivi.

- L’ultimo strato deve sempre essere di dimensionalità pari a quella delle classi del dataset, in modo che il modello associ un output corrispondente alla sua classificazione

il modello viene poi compilato, e scelta la funzione di loss (la funzione di costo) che permette al modello di trovare la miglior strada per “imparare” e aggiustare i pesi nella fase di backward propagation.



in questa immagine vengono visualizzati i vari strati del modello con le shape in output e i parametri che si creano nei vari strati

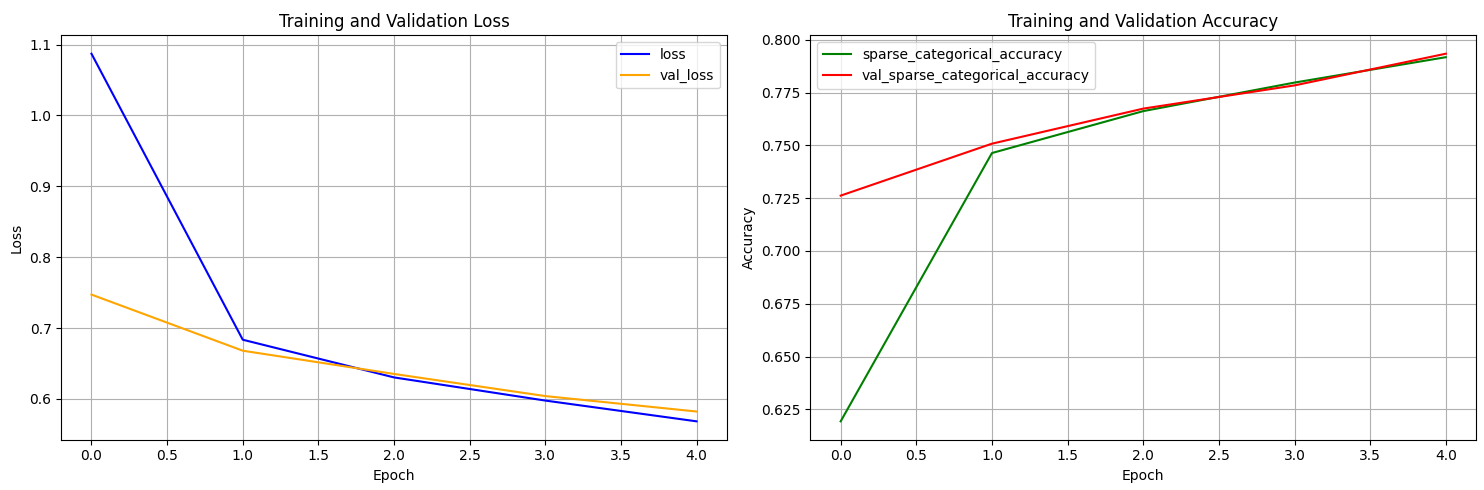
Successivamente viene fatto l’effettivo allenamento del modello con la funzione .fit

aggiungendo al callback (azioni compiute durante le epoche di addestramento) qualche statistica di come sta performando.

La classe TrainingMonitor analizza l’accuracy del training e della validation oltre che la loss function, per capire se il modello sta andando in overfitting (la train accuracy aumenta ma la validation non lo segue, impara a memoria) o underfitting

inoltre è presente un early stopping, che capisce quando il modello non sta migliorando e ferma le epoche per evitare computazione inutile.

Vengono visualizzati poi i grafici dell’allenamento



e l’inferenza da come risultati: 20/25 corrette predizioni



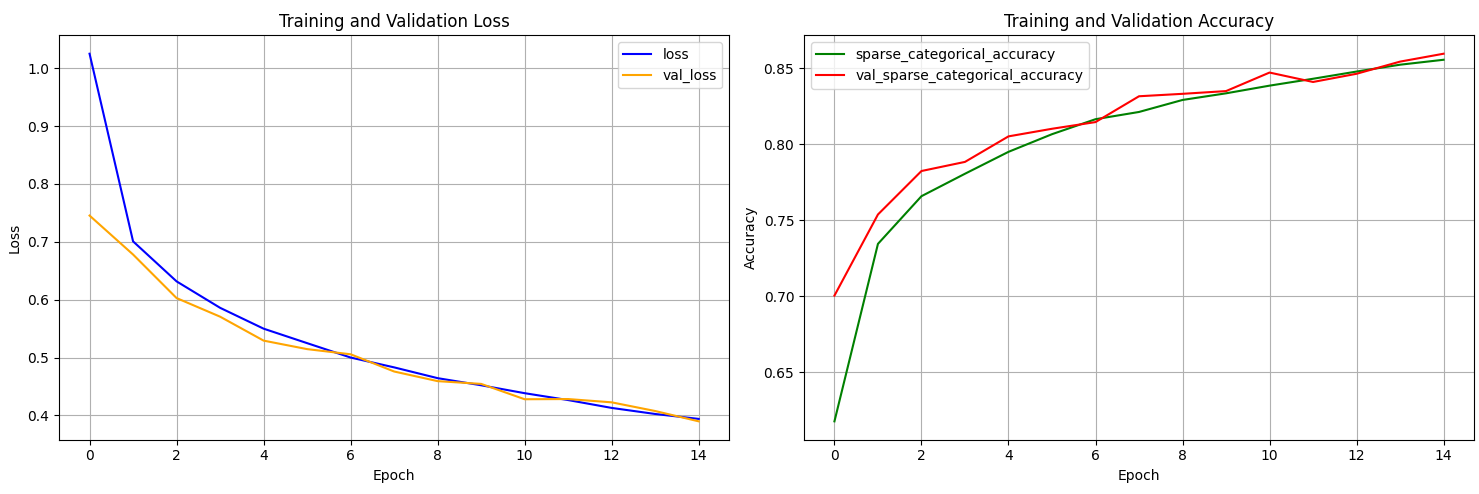
Lo script crea\_modello\_2 invece migliora il modello aggiungendo:

maggiori filtri convoluzionali nel secondo strato, dovrebbe estrarre più caratteristiche dalle immagini

una seconda convoluzione e un altro max pooling,

ulteriori strati dense come penultimo strato

l’addestramento del modello migliorato è:

l’inferenza sul modello migliorato restituisce: 24/25 predizioni corrette

