Progetto: Digit recognition

Programmazione di Applicazioni Data Intensive
Laurea in Ingegneria e Scienze Informatiche
DISI - Università di Bologna, Cesena

Andrea Micheli `andrea.micheli3@studio.unibo.it` matr. 000843618

Setup

- Installare le seguenti librerie necessarie al funzionamento del progetto, in caso sistema:
  - **Tensorflow** utilizzato come backend per la computazione da utilizzare attr
  - **Wget** utilizzato per ottenere FILES/DIR dal Web.

In [1]:
```
%pip install tensorflow
%pip install wget
```

```
Requirement already satisfied: tensorflow in d:\anaconda3\lib\site-packages (2.9.1)
Requirement already satisfied: keras-preprocessing>=1.1.1 in d:\anaconda3\lib\site-packages (from tenso
Requirement already satisfied: six>=1.12.0 in d:\anaconda3\lib\site-packages (from tensorflow) (1.15.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in d:\anaconda3\lib\site-packages (from tensorflow)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in d:\anaconda3\lib\site-packages (from tensorflow
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in d:\anaconda3\lib\site-packages (
Requirement already satisfied: setuptools in d:\anaconda3\lib\site-packages (from tensorflow) (50.3.1.p
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in d:\anaconda3\lib\site-packages (from tensorfl
Requirement already satisfied: grpcio<2.0,>=1.24.3 in d:\anaconda3\lib\site-packages (from tensorflow)
Requirement already satisfied: libclang>=13.0.0 in d:\anaconda3\lib\site-packages (from tensorflow) (14
Requirement already satisfied: google-pasta>=0.1.1 in d:\anaconda3\lib\site-packages (from tensorflow)
Requirement already satisfied: absl-py>=1.0.0 in d:\anaconda3\lib\site-packages (from tensorflow) (1.0.
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in d:\anaconda3\lib\site-packages
Requirement already satisfied: tensorboard<2.10,>=2.9 in d:\anaconda3\lib\site-packages (from tensorflo
Requirement already satisfied: flatbuffers<2,>=1.12 in d:\anaconda3\lib\site-packages (from tensorflow)
Requirement already satisfied: numpy>=1.20 in d:\anaconda3\lib\site-packages (from tensorflow) (1.22.4)
Requirement already satisfied: termcolor>=1.1.0 in d:\anaconda3\lib\site-packages (from tensorflow) (1.
Requirement already satisfied: typing-extensions>=3.6.6 in d:\anaconda3\lib\site-packages (from tensorf
Requirement already satisfied: packaging in d:\anaconda3\lib\site-packages (from tensorflow) (20.4)
Requirement already satisfied: wrapt>=1.11.0 in d:\anaconda3\lib\site-packages (from tensorflow) (1.11.
Requirement already satisfied: h5py>=2.9.0 in d:\anaconda3\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: astunparse>=1.6.0 in d:\anaconda3\lib\site-packages (from tensorflow) (1
Requirement already satisfied: opt-einsum>=2.3.2 in d:\anaconda3\lib\site-packages (from tensorflow) (3
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in d:\anaconda3\lib\site-packages (from te
(1.8.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in d:\anaconda3\lib\site-packages (from tensorboar
Requirement already satisfied: requests<3,>=2.21.0 in d:\anaconda3\lib\site-packages (from tensorboard<
Requirement already satisfied: wheel>=0.26 in d:\anaconda3\lib\site-packages (from tensorboard<2.10,>=2
Requirement already satisfied: markdown>=2.6.8 in d:\anaconda3\lib\site-packages (from tensorboard<2.10
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in d:\anaconda3\lib\site-packages (from
w) (0.4.6)
Requirement already satisfied: werkzeug>=1.0.1 in d:\anaconda3\lib\site-packages (from tensorboard<2.10
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in d:\anaconda3\lib\site-packages
rflow) (0.6.1)
Requirement already satisfied: pyparsing>=2.0.2 in d:\anaconda3\lib\site-packages (from packaging->tens
Requirement already satisfied: pyasn1-modules>=0.2.1 in d:\anaconda3\lib\site-packages (from google-aut
->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in d:\anaconda3\lib\site-packages
rboard<2.10,>=2.9->tensorflow) (4.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in d:\anaconda3\lib\site-packages (from google-au
9->tensorflow) (5.2.0)
Requirement already satisfied: charset-normalizer~=2.0.0; python_version >= "3" in d:\anaconda3\lib\sit
0->tensorboard<2.10,>=2.9->tensorflow) (2.0.12)
Requirement already satisfied: idna<4,>=2.5; python_version >= "3" in d:\anaconda3\lib\site-packages (f
d<2.10,>=2.9->tensorflow) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda3\lib\site-packages (from requests<3,>=
sorflow) (2020.6.20)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in d:\anaconda3\lib\site-packages (from requests<3
tensorflow) (1.25.11)
Requirement already satisfied: importlib-metadata>=4.4; python_version < "3.10" in d:\anaconda3\lib\sit
ensorboard<2.10,>=2.9->tensorflow) (4.11.4)
Requirement already satisfied: requests-oauthlib>=0.7.0 in d:\anaconda3\lib\site-packages (from google-
```

```
ard<2.10,>=2.9->tensorflow) (1.3.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in d:\anaconda3\lib\site-packages (from pyasn1-modu
tensorboard<2.10,>=2.9->tensorflow) (0.4.8)
Requirement already satisfied: zipp>=0.5 in d:\anaconda3\lib\site-packages (from importlib-metadata>=4.
n>=2.6.8->tensorboard<2.10,>=2.9->tensorflow) (3.4.0)
Requirement already satisfied: oauthlib>=3.0.0 in d:\anaconda3\lib\site-packages (from requests-oauthli
>=0.4.1->tensorboard<2.10,>=2.9->tensorflow) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: wget in d:\anaconda3\lib\site-packages (3.2)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import tensorflow as tf
```

- È consigliato eseguire il seguente progetto utilizzando la GPU per accelerare i
- Eseguire il seguente comando per accertarsi che la GPU sia vista correttament

  es. ['/device:CPU:0', '/device:GPU:0']

In [3]:
```
from tensorflow.python.client import device_lib

def get_available_devices():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos]

print(get_available_devices())
```

```
['/device:CPU:0', '/device:GPU:0']
```

- Librerie di uso comune necessarie alla rappresentazione o caricamento dei dat

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Descrizione del problema

- ## Obiettivo: Classificare correttamente numeri s
  - Si vuole realizzare una NN in grado di classificare automaticamente nume percentuale di errore più bassa possibile.
  - MNIST (Modified National Institute of Standards and Technology) è un dat presenti 60000 esempi di training e 10000 esempi di test da utilizzare per

proprio modello.
- Le immagini all'interno del MNIST sono rappresentazioni di cifre scritte a m seperiori ed impiegati d'ufficio.
- Le cifre presenti all'interno del MNIST vanno da 0 a 9.

Funzioni utili per la raccolta dati

- **Download** utilizzato per scaricare FILE/DIR nella cartella corrente in caso non

In [5]:
```
def download(file, url):
    import os
    import wget

    # controlla se il file/dir è già presente all'interno della cartella corrente
    if not os.path.isfile(file):
        # scarica il file/dir richiesta dall'URL specificato
        wget.download(url+file)
```

- **Remove** utilizzato per eliminare i FILE/DIR una volta generato il file .csv.

In [6]:
```
def remove(file):
    import os

    # controlla se nella cartella corrente esiste il file
    # che si vuole eliminare
    if os.path.exists(file):
        # rimuove file
        os.remove(file)
```

- **Unzip** utilizzato per decomprimere i quattro insiemi del MNIST.

In [7]:
```
def unzip(file, file_type, nth):
    import gzip
    import shutil

    # rimuove l'estensione della cartella compressa
    file_name = file.split(".")[0]
    # aggiorna il file name con il tipo di estensione del file corrente
    file_name = file_name.replace('-', ".", nth)
    file_name = file_name.replace('.', "-", nth-1)

    # decomprime la cartella e copia il suo contenuto nella cartella corrente
    with gzip.open(file, 'rb') as f_in:
        with open(file_name, 'wb') as f_out:
            shutil.copyfileobj(f_in, f_out)
    return file_name
```

- **Convert** utilizzato per convertire i file .idx3-ubyte del dataset in file .csv.

In [8]:
```
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    # ricopia riga per riga tutte le immagini presenti nel dataset
    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()
```

- **MNIST_CSV** utilizza tutte le funzioni precedentemente definite. Se all'interno d sono presenti i file .csv del training e validation set, la funzione seguente scaric necessari all'utilizzo dei dati tramite pandas.

In [9]:
```
def mnist_csv(file_images, file_labels, url, csv_name, size, file_type, rmv):
    import os

    # controlla se il file .csv che vuole essere
    # generato esiste già nella cartella corrente
    if not os.path.isfile(csv_name):
        # scarica il set di immagini
        download(file_images, url)
        # scarica il set di label di ogni immagine
        download(file_labels, url)
        file_name_images = unzip(file_images, file_type, 2)
        file_name_labels = unzip(file_labels, file_type, 2)
        # concatena il set di immagini e label in un singolo file .csv
        convert(file_name_images, file_name_labels, csv_name, size)
        if rmv:
            remove(file_images)
            remove(file_labels)
            remove(file_name_images)
            remove(file_name_labels)
```

- Vengono scaricati gli insiemi di train e test del MNIST e poi generati i relativi .cs

In [10]:
```
url = "http://yann.lecun.com/exdb/mnist/"
file_type = '.idx3-ubyte'
train_images = "train-images-idx3-ubyte.gz"
train_labels = "train-labels-idx1-ubyte.gz"
test_images = "t10k-images-idx3-ubyte.gz"
test_labels = "t10k-labels-idx1-ubyte.gz"
mnist_train = "mnist_train.csv"
mnist_test = "mnist_test.csv"

mnist_csv(train_images, train_labels, url, mnist_train, 60000, file_type, rmv=Tru
mnist_csv(test_images, test_labels, url, mnist_test, 10000, file_type, rmv=True)
```

Analisi esplorativa dei dati

- Generati i file .csv è possibile accedere ai dati utilizzando Pandas.

In [11]:
```
train_set = pd.read_csv(mnist_train, header=None)
val_set = pd.read_csv(mnist_test, header=None)
train_set.columns = val_set.columns = ["label"] + [f"pixel_{x}" for x in range(tr
```

- È ora possibile visualizzare le dimensioni dei due dataset.

In [12]:
```
print("Size of training data: ", train_set.shape)
print("Size of test data: ", val_set.shape)
```

```
Size of training data:  (60000, 785)
Size of test data:  (10000, 785)
```

- Pandas facilita la rappresentazione dei dati sotto forma di tabelle.
- La colonna **label**, la prima colonna della tabella, rappresenta la corretta classifi rappresentato sotto forma di immagine.
- Le restanti 784 colonne rappresentano il valore di ogni pixel (da 0 a 255, da ne di una immagine 28x28 pixel.

In [13]:
```
train_set.head(10)
```

	label	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	...	pix
0	5	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	...	0
2	4	0	0	0	0	0	0	0	0	0	...	0
3	1	0	0	0	0	0	0	0	0	0	...	0
4	9	0	0	0	0	0	0	0	0	0	...	0
5	2	0	0	0	0	0	0	0	0	0	...	0
6	1	0	0	0	0	0	0	0	0	0	...	0
7	3	0	0	0	0	0	0	0	0	0	...	0
8	1	0	0	0	0	0	0	0	0	0	...	0
9	4	0	0	0	0	0	0	0	0	0	...	0

10 rows × 785 columns

In [14]:
```
val_set.head(10)
```

	label	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	...	pix
0	7	0	0	0	0	0	0	0	0	0	...	0
1	2	0	0	0	0	0	0	0	0	0	...	0
2	1	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	4	0	0	0	0	0	0	0	0	0	...	0
5	1	0	0	0	0	0	0	0	0	0	...	0
6	4	0	0	0	0	0	0	0	0	0	...	0
7	9	0	0	0	0	0	0	0	0	0	...	0
8	5	0	0	0	0	0	0	0	0	0	...	0
9	9	0	0	0	0	0	0	0	0	0	...	0

10 rows × 785 columns

- Le immagini all'interno del MNIST vengono, per lo più, centrate all'interno della
- Per poter meglio analizzare l'intervallo di numeri che ogni pixel può rappresenta osservare colonne centrali.

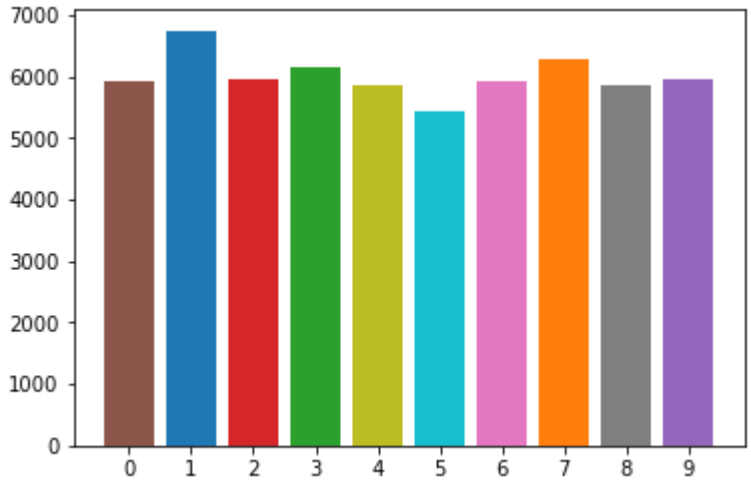
In [15]:
```
train_set.iloc[:,160:210].head(8)
```

	pixel_159	pixel_160	pixel_161	pixel_162	pixel_163	pixel_164	pixel_165	pixel_166	pixel
0	26	166	255	247	127	0	0	0	0
1	237	0	0	0	0	0	0	0	0
2	0	67	232	39	0	0	0	0	0
3	253	255	63	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	7	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	253	174	6	0	0	0	0	0	0

8 rows × 50 columns

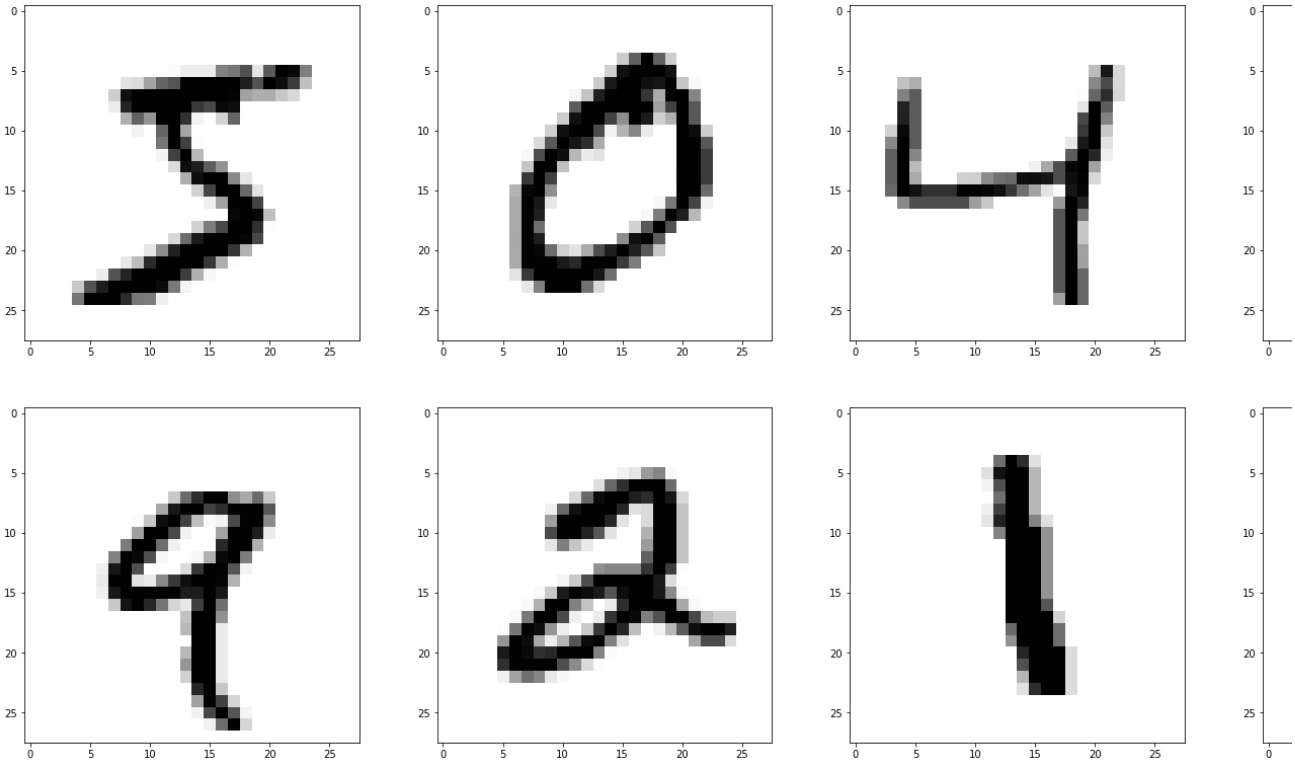
- Visualizziamo la distribuzione dei valori suddivisi in classi.

In [16]:
```
counts = train_set["label"].value_counts()
label = []

for i in range(len(counts.index)):
    # aggiungiamo una nuova barra
    plt.bar(counts.index[i], counts.values[i])

    # aggiungiamo l'indice corrente
    label.append(i)
    plt.xticks(range(len(label)), labels=label)
plt.show()
```



- Per poter rappresentare i numeri sotto forma di immagini è necessaria una rimo
- Ogni riga del dataset viene rimodellata in una matrice 28 pixel x 28 pixel.
- Fatto ciò è possibile visualizzare l'immagine relativa ad ogni riga presente all'in

In [17]:
```
plt.figure(figsize=(28, 28))
# mostriamo sono i primi 8 numeri presenti nel dataset
for x in range(8):
    # dimensione massima del subplot 4x4 (16 numeri rappresentabili al massimo)
    plt.subplot(4, 4, x+1)
    plt.imshow(np.reshape(train_set.iloc[x, 1:].to_numpy(), (28,28)), cmap=plt.cm
plt.show()
```



Suddivisione dei dati

- Nel caso del MNIST database non è necessaria la separazione tra insiemei di t due set vengono già forniti separatamente.
- Selezionare i dati su cui lavorare:

- la variabile `X_train` contiene i valore di tutti i pixel che formano l'immagine
- la variabile `y_train` da predirre è il numero rappresentato nell'immagine (d

In [18]:
```
X_train = train_set.drop(labels = ["label"], axis = 1)
y_train = train_set["label"]
```

In [19]:
```
# viene scelta una sezione centrale per mostrare
# meglio la normalizzazione che verrà eseguita
X_train.iloc[:,160:210].head(8)
```

	pixel_160	pixel_161	pixel_162	pixel_163	pixel_164	pixel_165	pixel_166	pixel_167	pixel
0	166	255	247	127	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	67	232	39	0	0	0	0	0	0
3	255	63	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	174	6	0	0	0	0	0	0	0

8 rows × 50 columns

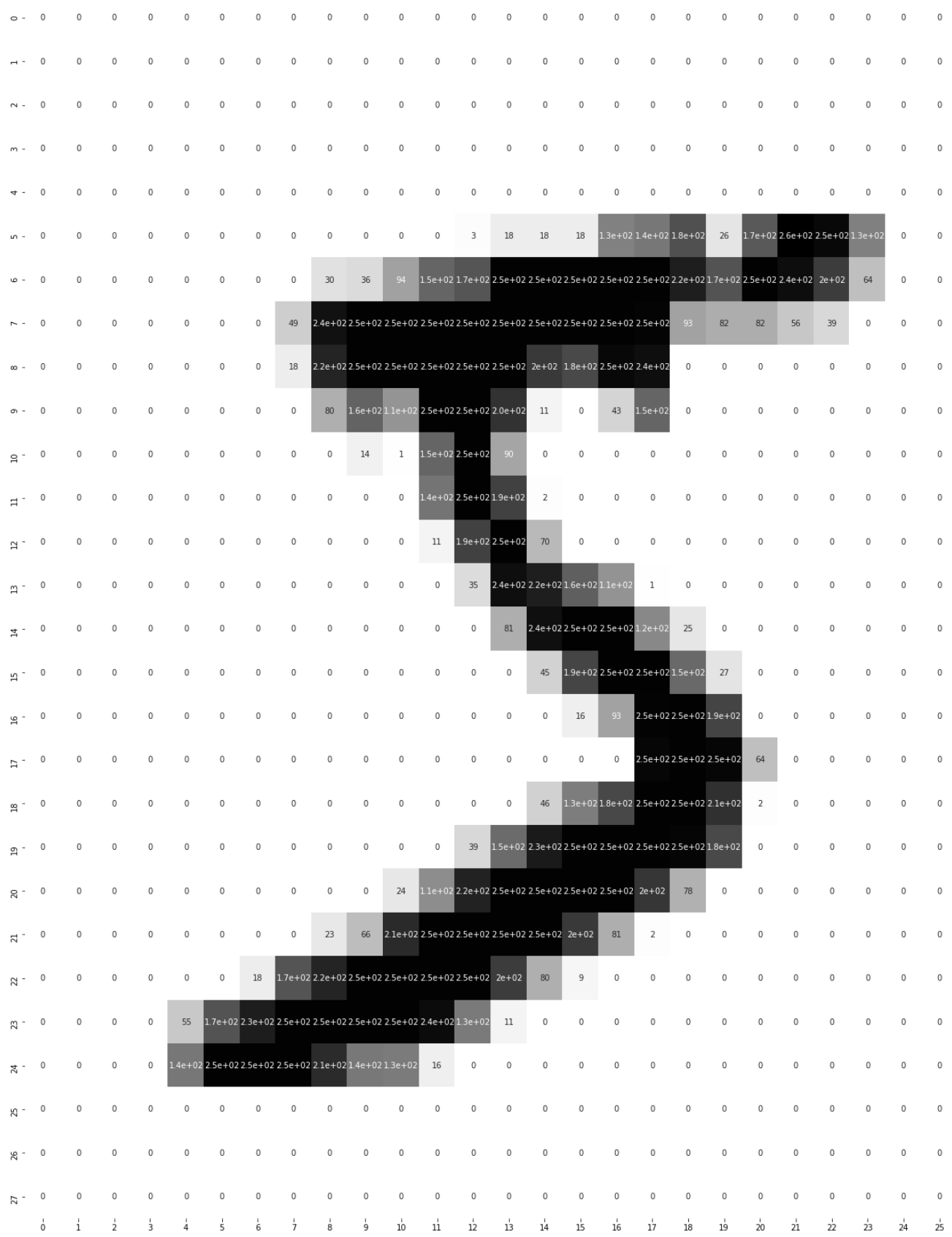
In [20]:
```
y_train.head(8)
```

```
0    5
1    0
2    4
3    1
4    9
5    2
6    1
7    3
Name: label, dtype: int64
```

Normalizzazione dei dati

- Per quale motivo è necessaria la normalizzazione del valore di ogni pixel?
  - Per aiutare a capire la necessità di una normalizzazione dei valori ho utiliz
  - Utilizzando la funzione heatmap della libreria è possibile visualizzare il valo dell'immagine.

In [21]:
```
import seaborn as sns
plt.figure(figsize=(28, 28))
sns.heatmap(np.reshape(X_train.iloc[0, :].to_numpy(), (28, 28)), annot=True, cmap
```



- Essendo ogni pixel un valore compreso tra 0 e 255, per normalizzare i nostri da
  dividere ogni singolo valore per 255.
- Questo passaggio viene eseguito perchè, valori di grandi dimensioni, potrebbe
  della NN più complicata.

In [22]:
```
X_train = X_train / 255.0
```

- Come si può osservare di seguito, eseguita la normalizzazione, i valori di ogni p

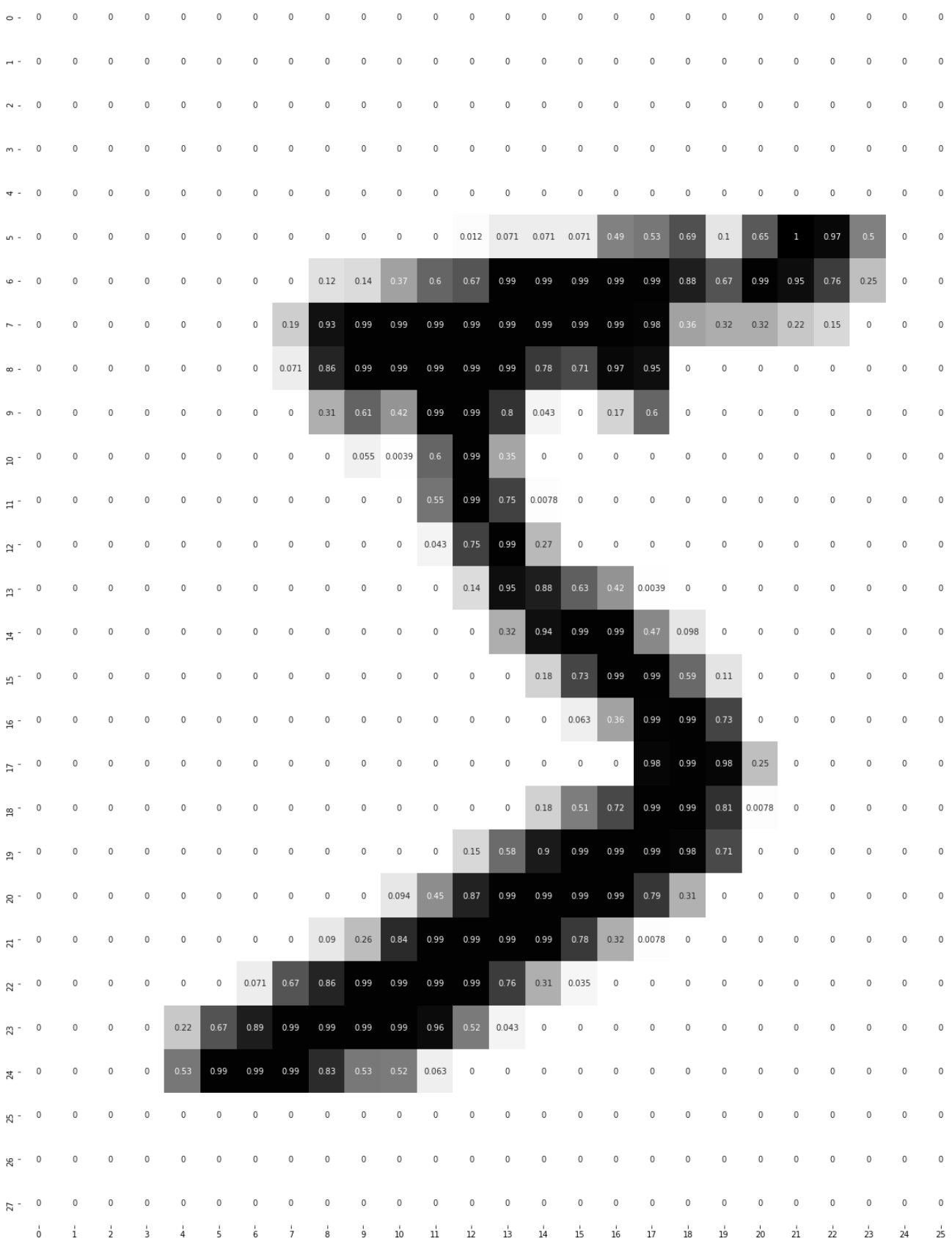
In [23]:
```
# viene scelta una sezione centrale per mostrare
# meglio la normalizzazione eseguita
X_train.iloc[:,160:210].head(8)
```

	pixel_160	pixel_161	pixel_162	pixel_163	pixel_164	pixel_165	pixel_166	pixel_167	pixel
0	0.650980	1.000000	0.968627	0.498039	0.0	0.0	0.0	0.0	0.0
1	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
2	0.262745	0.909804	0.152941	0.000000	0.0	0.0	0.0	0.0	0.0
3	1.000000	0.247059	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
4	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
5	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
6	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
7	0.682353	0.023529	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0

8 rows × 50 columns

- Analiziamo nuovamente l'heatmap fornita da Seaborn, per visualizzare le modi

In [24]:
```
plt.figure(figsize=(28, 28))
sns.heatmap(np.reshape(X_train.iloc[0, :].to_numpy(), (28, 28)), annot=True, cmap
```



- Eseguiamo le stesse operazioni sull'insieme di test.

In [25]:
```
X_val = val_set.drop(labels = ["label"], axis = 1)
y_val = val_set["label"]
```

In [26]:
```
X_val = X_val / 255.0
```

In [27]:
```
X_val.head(8)
```

	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9	...	p
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0

8 rows × 784 columns

In [28]:
```
y_val.head(8)
```

```
0    7
1    2
2    1
3    0
4    4
5    1
6    4
7    9
Name: label, dtype: int64
```

Rete neurale di classificazione

- Si vuole addestrare un modello di classificazione in grado di distinguere le conf formano ogni numero da 0 a 9.
- Per poter generare la rete neurale ho usato un modello di Keras demonimato `s`
- Il modello `Sequential` permette di aggiungere semplicemente strato su strato: c input **tensor** e un ouput **tensor**.

In [29]:
```
from tensorflow.keras.models import Sequential

model = Sequential()
```

- # Strato Dense
  - **Dense** è uno degli strati più utilizzati delle reti neurali.
  - Le operazioni che esegue questo strato sono:

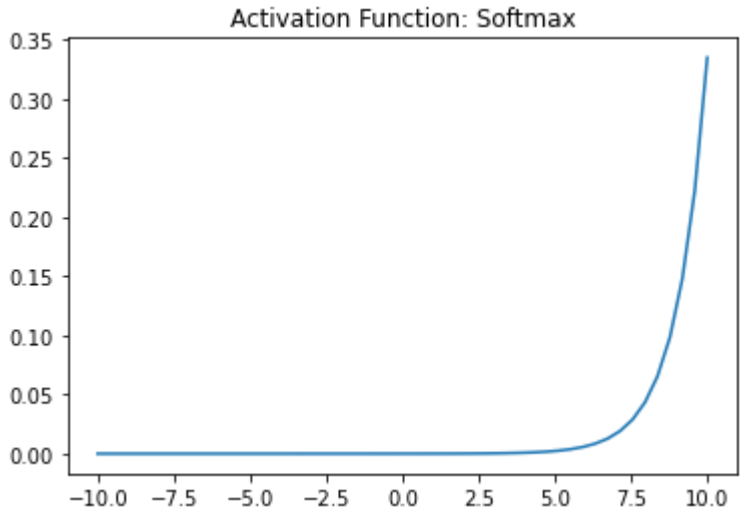
    `output = activation(dot(input, kernel) + bias)`

    dove activation è la funzione eseguita elemento per elemento, kernel una r strato e bias il vettore di pesi creati dallo strato (applicabile se `use_bias` è
  - Inoltre sul primo strato **Dense** bisogna indicare con `input_dim` il numero di
  - Per creare uno strato **Dense** bisogna specificare il numero nodi di output e utilizzare, cioè `activation` .

In [30]:
```
from tensorflow.keras.layers import Dense

model.add(Dense(10, activation="softmax", input_dim=X_train.shape[1]))
```

- La funzione di attivazione `softmax` converte un vettore di valori in una distribuzi determinare quale sia il numero che, in questo caso, la rete neurale pensa sia r

In [31]:
```
def softmax_visualizer(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)

x = np.linspace(-10, 10)
plt.plot(x, softmax_visualizer(x))
plt.axis('tight')
plt.title('Activation Function: Softmax')
plt.show()
```



- Per poter visualizzare il riepilogo della rete neurale si può utilizzare la funzione mostra la configurazione strato per strato.

In [32]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                7850

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

- Per compilare il modello è necessario specificare quali metriche sia necessario giudicarne le prestazioni.
- È necessario specificare:

- l'algoritmo di ottimizzazione da utilizzare.
- la misura d'errore.
- la metrica dell'errore da ottimizzare.

- Per questo progetto ho utilizzato `Adam` (metodo di discesa del gradiente stocas ottimizzazione, `Sparse Categorical Crossentropy` (utilizzata quando sono prese classificare) come misura d'errore e in aggiunta si vuole mostrare anche l'accu

In [33]:
```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

- Viene addestrato il modello per 5 epoche su `X_train` e `y_train`.

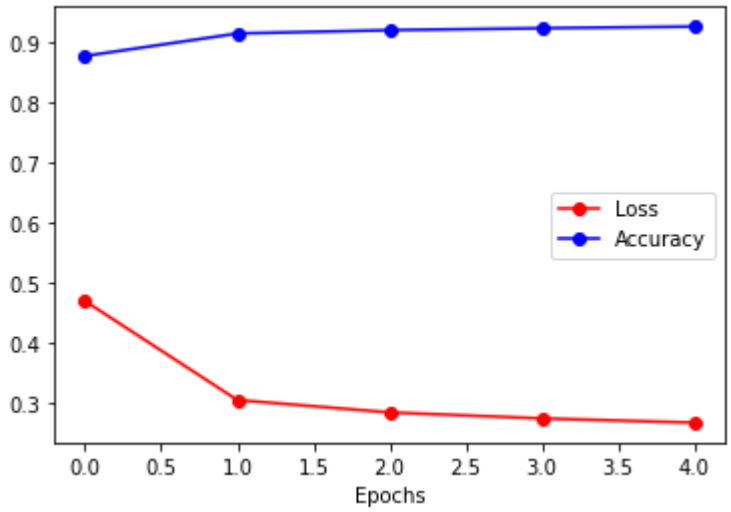
In [34]:
```
fit_history = model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 1ms/step - loss: 0.4692 - accuracy: 0.8777
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3041 - accuracy: 0.9153
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2834 - accuracy: 0.9208
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2734 - accuracy: 0.9241
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2664 - accuracy: 0.9269
```

La funzione **plot_loss_accuracy** serve per visualizzare la rappresentazione su gra `accuracy` ad ogni epoca.

In [35]:
```
def plot_loss_accuracy(model_history):
    plt.plot(model_history.history["loss"], "ro-")
    plt.plot(model_history.history["accuracy"], "bo-")
    plt.legend(["Loss", "Accuracy"])
    plt.xlabel("Epochs");
```

In [36]:
```
plot_loss_accuracy(fit_history)
```



- Tramite la funzione `predict` di **Keras** è possibile generare un vettore di probab classificare.

In [37]:
```
y_pred = model.predict(X_val)
```

```
313/313 [==============================] - 0s 640us/step
```

- La funzione `evaluate` permette di ottenere il valore di loss e l'accuratezza del n

In [38]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.2656 - accuracy: 0.9271
```

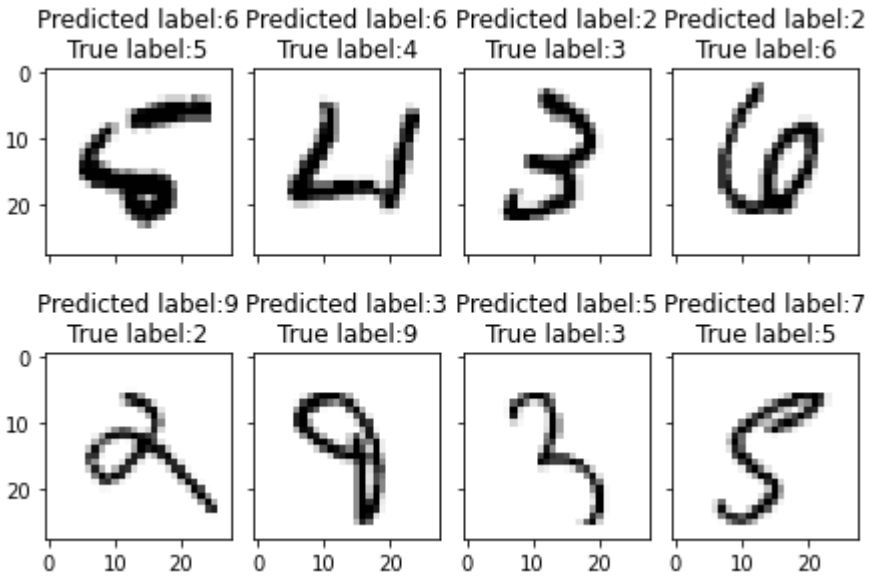
```
[0.26555007696151733, 0.9271000027656555]
```

La funzione **display_errors** permette di visualizzare in quali casi la rete neurale abl

classificazione dell'immagine. Oltre a mostrare la figura del numero errato, presnta, relativa etichetta predetta e quella reale.

In [39]:
```
def display_errors(label_set, predictions, n):
    x = 0; y = 0
    cols = round(n/2)
    rows = round(n/cols)
    _, axs = plt.subplots(rows, cols, sharex=True, sharey=True,constrained_layout
    for e in range(len(predictions)):
        if y >= rows:
            return
        if np.argmax(predictions[e]) != label_set[e]:
            axs[y,x].imshow(np.reshape(val_set.iloc[:, 1:].to_numpy()[e], (28,28)
            axs[y,x].set_title("Predicted label:{}\nTrue label:{}".format(np.argm
            x = 0 if x == cols-1 else x+1
            y = y+1 if x == 0 else y
```

In [40]:
```
display_errors(y_val, y_pred, 8)
plt.show();
```



- Eseguiamo ora una rimodellazione dei dati, traformando ogni riga in una matric
- In questo modo `X_train` non sarà più accessibile come un DataFrame Pandas altri strati di **Keras**, più appropriati al nostro tipo di input.

In [41]:
```
X_train = X_train.values.reshape(-1, 28, 28, 1)
X_val = X_val.values.reshape(-1, 28, 28, 1)
img_dim = X_train.shape[1]
```

- # Strato Flatten
  - Lo strato `Flatten` permette di convertire l'input ed "appiattirlo", rimodellanc monodimensionale.
  - In questo modo è possibile aggiornare il modello, affiché sia conforme alla

In [42]:
```
from tensorflow.keras.layers import Flatten

model = Sequential([
    Flatten(input_shape=(img_dim, img_dim, 1)),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

- Rieseguiamo ora i test per verificare che il modello ottenga più o meno lo stess

In [43]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_1 (Dense)             (None, 10)                7850

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
fit_history = model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4699 - accuracy: 0.8778
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3044 - accuracy: 0.9149
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2835 - accuracy: 0.9205
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2732 - accuracy: 0.9244
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2666 - accuracy: 0.9255
```

In [45]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.2699 - accuracy: 0.9251
```

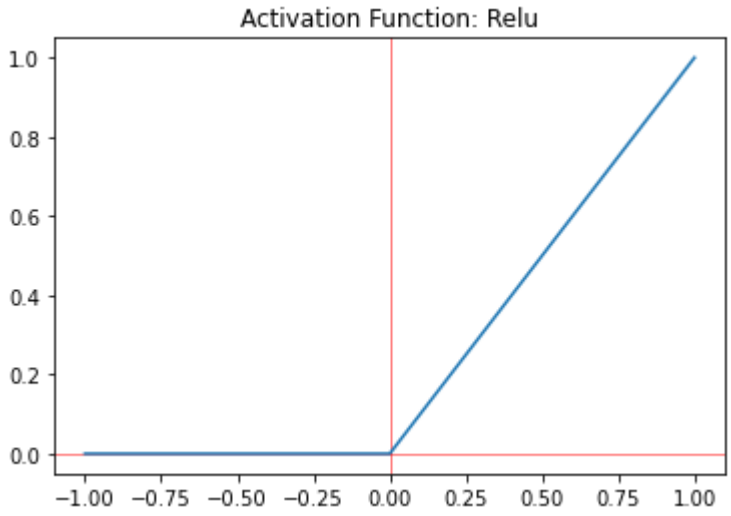
```
[0.2698509097099304, 0.9251000285148621]
```

- # Hidden Layers
  - Aggiungiamo ulteriori strati alla rete neurale, per migliorarne l'accuracy.
  - L'aggiunta di ulteriori strati permette alla rete neurale di rinoscere familiarit correttamente dataset con correlazione tra i valori più complicate.

In [46]:
```
model = Sequential([
    Flatten(input_shape=(img_dim, img_dim, 1)),
    Dense(32, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

- La funzione di attivazione **relu** controlla elemento per elemento: in caso il valor stesso, altrimenti viene impostato a 0.

In [47]:
```
def relu_visualizer(x):
    return np.maximum(0, x)

x = np.arange(-1, 1.1, 0.1)
y = [relu_visualizer(i) for i in x]
plt.axvline(x=0, color="red", linewidth=.5)
plt.axhline(y=0, color="red", linewidth=.5)
plt.title('Activation Function: Relu')
plt.plot(x, y)
plt.show()
```



- Si può osservare come il numero di parametri cresca drasticamente.
- Il numero totale di parametri del modello si può calcolare come:

```
n_params = input x layers_shape + layer_shape
```

In [48]:
```
784*32 + 32 + 32*10 + 10
```

```
25450
```

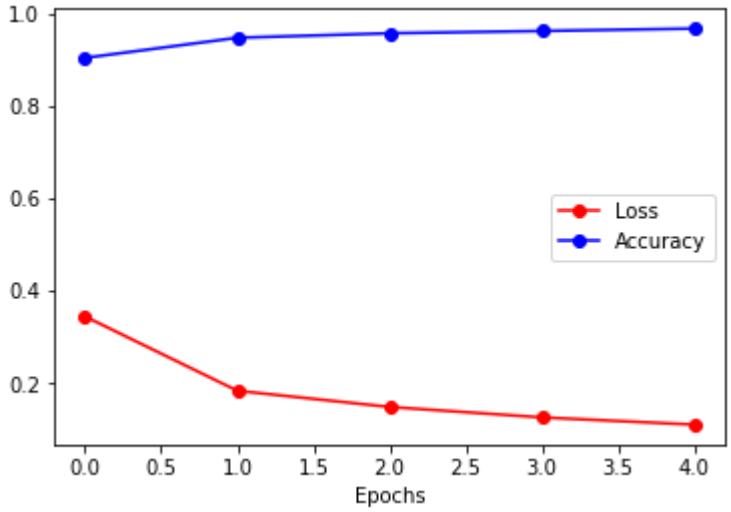
In [49]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 32)                25120

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
fit_history = model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3443 - accuracy: 0.9042
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1833 - accuracy: 0.9477
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1483 - accuracy: 0.9575
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1257 - accuracy: 0.9625
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1100 - accuracy: 0.9677
```

In [51]:
```
plot_loss_accuracy(fit_history)
```



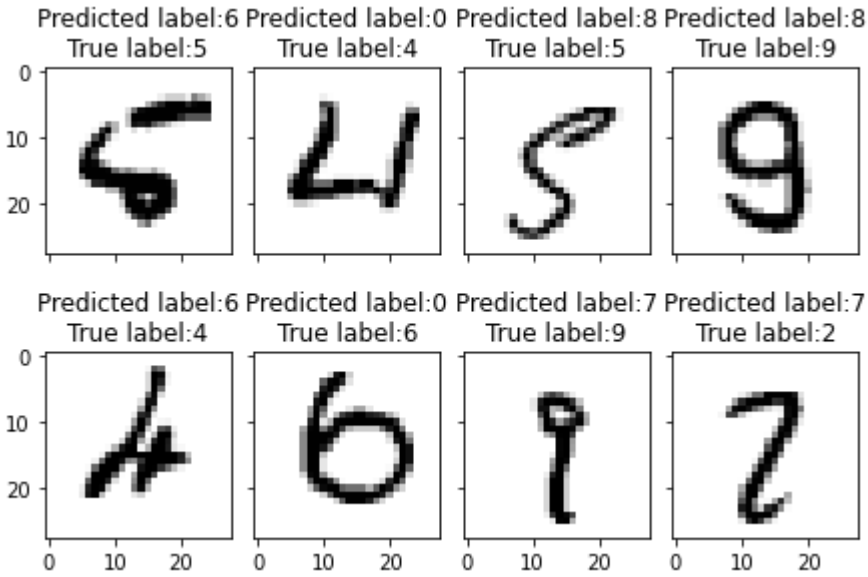
In [52]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1278 - accuracy: 0.9640

[0.1277984231710434, 0.9639999866485596]
```

- Con l'aggiunta di un ulteriore strato **Dense** l'architettura del modello corrente ha
rispetto al suo predecessore.
- Osserviamo i possibili nuovi casi in cui il modello compie errori di classificazion

In [53]:
```
y_pred = model.predict(X_val)

display_errors(y_val, y_pred, 8)
plt.show();
```

```
313/313 [==============================] - 0s 701us/step
```



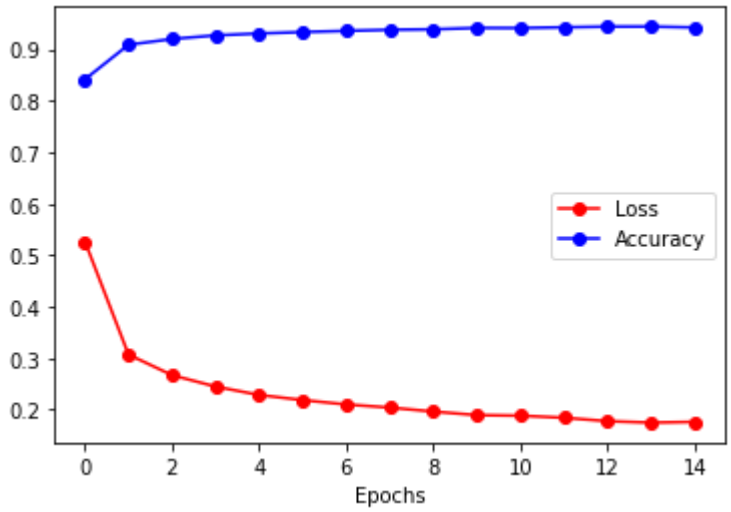
- # Strato Dropout
  - È essenziale alle rete neurali in quanto previene overfitting.
  - Durante la fase di addestramento, con una frequenza `rate` (come mostrat
    0.25), ignora nodi della rete disattivando le connessioni in entrata o in usci
  - Gli strati di **Dropout** sono attivi solo durante la fase di training del modello.

In [54]:
```
from tensorflow.keras.layers import Dropout

model = Sequential([
    Flatten(input_shape=(img_dim, img_dim, 1)),
    Dense(32, activation="relu"),
    Dropout(0.25),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

In [55]:
```
fit_history = model.fit(X_train, y_train, epochs=15)
```

```
Epoch 1/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5268 - accuracy: 0.8419
Epoch 2/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3072 - accuracy: 0.9090
Epoch 3/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2673 - accuracy: 0.9203
Epoch 4/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2450 - accuracy: 0.9272
Epoch 5/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2287 - accuracy: 0.9309
Epoch 6/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2188 - accuracy: 0.9337
Epoch 7/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2103 - accuracy: 0.9360
Epoch 8/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2043 - accuracy: 0.9381
Epoch 9/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1965 - accuracy: 0.9390
Epoch 10/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1896 - accuracy: 0.9417
Epoch 11/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1885 - accuracy: 0.9415
Epoch 12/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1845 - accuracy: 0.9430
Epoch 13/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1779 - accuracy: 0.9443
Epoch 14/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1750 - accuracy: 0.9444
Epoch 15/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1765 - accuracy: 0.9427
```

In [56]:
```
plot_loss_accuracy(fit_history)
```



- Come si può notare dai dati di loss ed accuracy sopra riportati, la fase di trainin di accuratezza inferiore rispetto al modello precente, pur avendo allenato il moc
- A differenza del valore di accuratezza ottenuto sull'input di training, quella riport risulta superiore al modello precedente.

In [57]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.1306 - accuracy: 0.9633

[0.13055264949798584, 0.9632999897003174]
```

- # Reti Convoluzionali
  - Per poter riconoscere pattern all'interno di immagini, indipendemente dal p utilizzano reti convoluzionali.
  - I primi strati convoluzionali permettono di identificare caratteristiche all'inte vengono date come training al modello. Man mano che i dati in input prose neurale, i pattern diventano sempre più esclusivi.
  - Nel mio caso l'utilizzo di strati convoluzionali permette alla rete neurale di i formano la rappresentazione grafica del numero.
  - I modelli di reti convoluzionali sono progettati per funzionare con immagini anche fungere da input monodimensioni e tridimensionali.

- # Strato Conv2D

- Lo strato **Conv2D** esegue una moltiplicazione, elemento per elemento, tra `kernel_size` dell'input e `filter` (insieme di pesi).
- La dimensione di `kernel_size` è volutamente più piccola dell'immagine di i possibile applicare su diverse porzioni dell'immagine gli stessi pesi.
- L'approccio di uno stesso insieme di pesi, che vengono utilizzati su tutta l'i poter riscontrare le stesse caratteristiche in diverse sezioni di essa.
- Riconoscere pattern senza preoccuparsi della loro posizione all'interno del neurale di aumentare drasticamente le sue prestazioni.

In [58]:
```
from tensorflow.keras.layers import Conv2D

model = Sequential([
    Conv2D(filters=64, kernel_size=(5, 5), activation="relu", input_shape=(img_di
    Dropout(0.25),
    Conv2D(filters=64, kernel_size=(3, 3), activation="relu"),
    Dropout(0.25),
    Flatten(),
    Dense(32, activation="relu"),
    Dropout(0.25),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

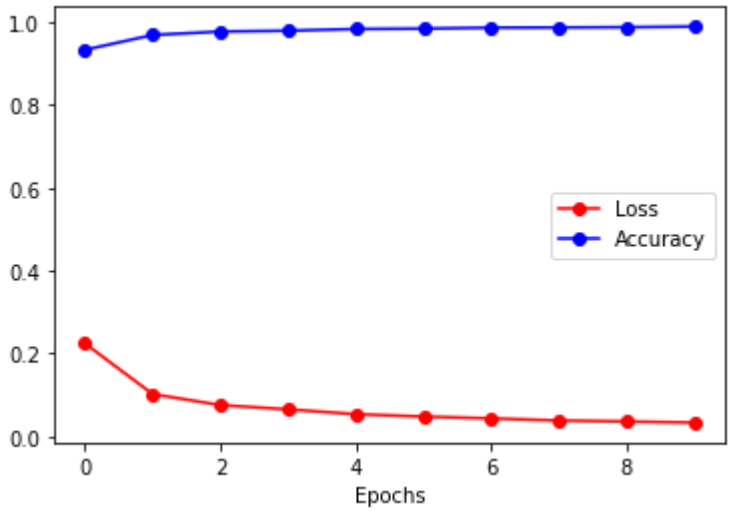
In [59]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 64)        1664

 dropout_1 (Dropout)         (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 22, 22, 64)        36928

 dropout_2 (Dropout)         (None, 22, 22, 64)        0

 flatten_3 (Flatten)         (None, 30976)             0

 dense_6 (Dense)             (None, 32)                991264

 dropout_3 (Dropout)         (None, 32)                0

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 1,030,186
Trainable params: 1,030,186
Non-trainable params: 0
_________________________________________________________________
```

In [60]:
```
fit_history = model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.2242 - accuracy: 0.9335
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1019 - accuracy: 0.9694
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0754 - accuracy: 0.9777
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0651 - accuracy: 0.9798
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0534 - accuracy: 0.9838
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0477 - accuracy: 0.9848
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0432 - accuracy: 0.9870
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0380 - accuracy: 0.9875
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0359 - accuracy: 0.9882
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0330 - accuracy: 0.9898
```

In [61]:
```
plot_loss_accuracy(fit_history)
```



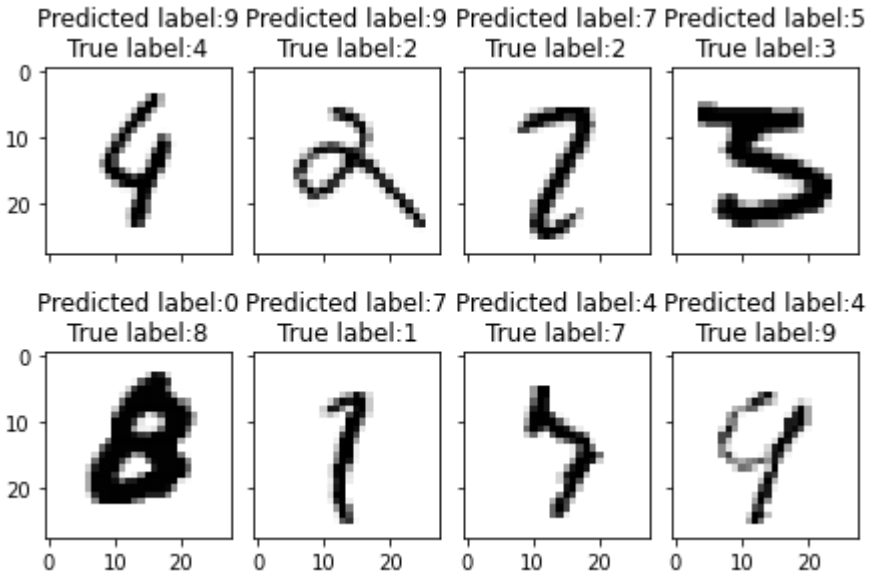
In [62]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0386 - accuracy: 0.9904

[0.03863911330699921, 0.9904000163078308]
```

In [63]:
```
y_pred = model.predict(X_val)
```

```
313/313 [==============================] - 0s 1ms/step
```

In [64]:
```
display_errors(y_val, y_pred, 8)
plt.show();
```



- # Strato MaxPooling
  - Per permettere alla rete convoluzionale di identificare le stesse caratteristi loro posizione all'interno dell'immagine, si utilizzano strati di **Pooling**.
  - In modo simile agli strati **Conv2D**, anche **Pooling** utilizza un filtro di dimen (nell'esempio successivo (2, 2)) che, posizionato sui valori in input ricevuti identifica, in questo caso, il maggiore tra i due valori e lo riporta in una ma
  - Tramite la variabile `strides` è possibile specificare la distanza per la quale

In [65]:
```
from tensorflow.keras.layers import MaxPooling2D

model = Sequential([
    Conv2D(filters=64, kernel_size=(5, 5), activation="relu", input_shape=(img_di
    Conv2D(filters=64, kernel_size=(3, 3), activation="relu"),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.25),
    Flatten(),
    Dense(32, activation="relu"),
    Dropout(0.25),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

In [66]:
```
fit_history = model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2354 - accuracy: 0.9282
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0996 - accuracy: 0.9698
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0759 - accuracy: 0.9767
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0628 - accuracy: 0.9804
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0556 - accuracy: 0.9830
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0466 - accuracy: 0.9847
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0411 - accuracy: 0.9865
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0394 - accuracy: 0.9874
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0338 - accuracy: 0.9895
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0330 - accuracy: 0.9891
```

In [67]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0351 - accuracy: 0.9919

[0.03505093604326248, 0.9919000267982483]
```

- # Strato BatchNormalization
  - Questo strato permette di normalizzare l'input in arrivo ad un altro strato.
  - Normalizzare l'output di ogni strato aiuta rete neurali di grandi dimensioni r diminuendo così il tempo richiesto per il training.

In [68]:
```
from tensorflow.keras.layers import BatchNormalization

model = Sequential([
    Conv2D(filters=64, kernel_size=(5, 5), activation="relu", input_shape=(img_di
    BatchNormalization(),
    Conv2D(filters=64, kernel_size=(3, 3), activation="relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.25),
    Flatten(),
    Dense(32, activation="relu"),
    BatchNormalization(),
    Dropout(0.25),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=[
```

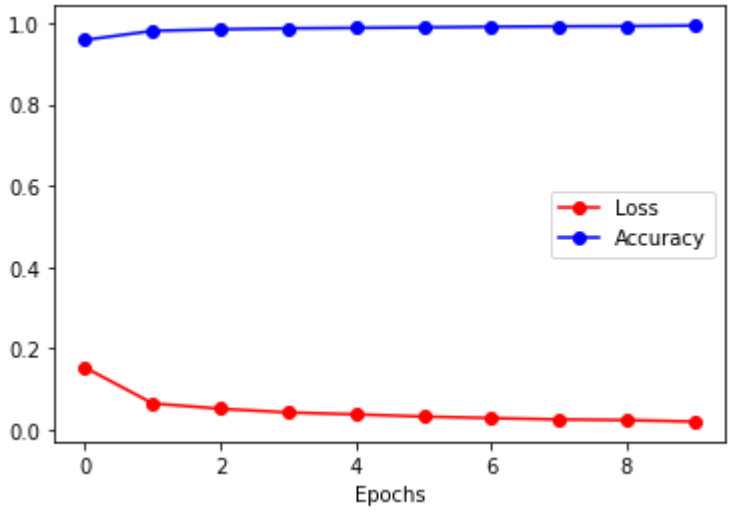
In [69]:
```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 24, 24, 64)        1664

 batch_normalization (BatchN  (None, 24, 24, 64)       256
 ormalization)

 conv2d_5 (Conv2D)           (None, 22, 22, 64)        36928

 batch_normalization_1 (Batc  (None, 22, 22, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 11, 11, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 11, 11, 64)        0

 flatten_5 (Flatten)         (None, 7744)              0

 dense_10 (Dense)            (None, 32)                247840

 batch_normalization_2 (Batc  (None, 32)               128
 hNormalization)

 dropout_7 (Dropout)         (None, 32)                0

 dense_11 (Dense)            (None, 10)                330

=================================================================
Total params: 287,402
Trainable params: 287,082
Non-trainable params: 320
_________________________________________________________________
```

In [70]:
```
fit_history = model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.1525 - accuracy: 0.9588
Epoch 2/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0639 - accuracy: 0.9811
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0509 - accuracy: 0.9849
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0417 - accuracy: 0.9872
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0370 - accuracy: 0.9884
Epoch 6/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0316 - accuracy: 0.9897
Epoch 7/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.0279 - accuracy: 0.9912
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0246 - accuracy: 0.9921
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0230 - accuracy: 0.9928
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0192 - accuracy: 0.9942
```

In [71]:
```
plot_loss_accuracy(fit_history)
```



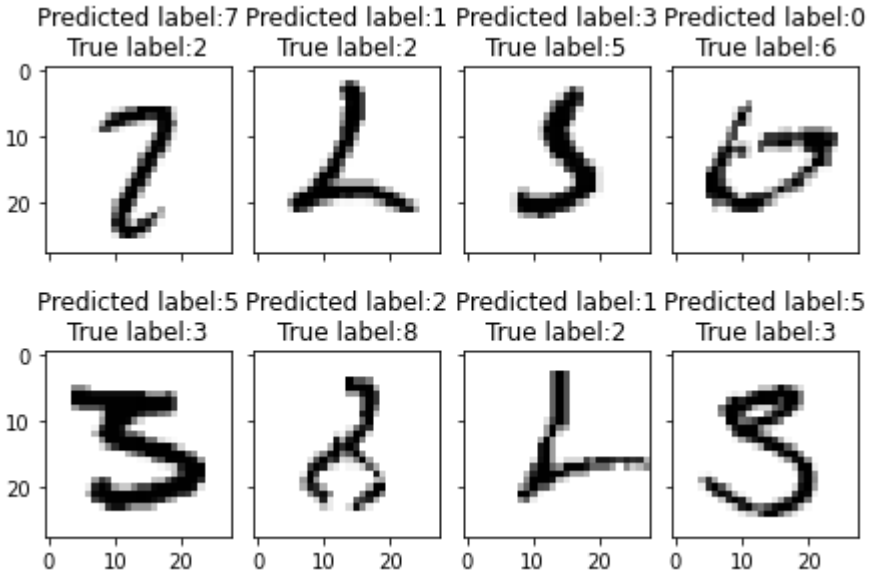
In [72]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0248 - accuracy: 0.9920

[0.024780577048659325, 0.9919999837875366]
```

In [73]:
```
y_pred = model.predict(X_val)
```

```
313/313 [==============================] - 0s 1ms/step
```

In [74]:
```
display_errors(y_val, y_pred, 8)
plt.show();
```



Keras Tuner

- Framework utile nella ricerca dei migliori iperparametri di una rete neurale.
- Facilita la ricerca di nuovi modelli, proponendo un'analisi per iterazioni delle mi relativi iperparametri.

In [75]:
```
%pip install -U keras-tuner
```

```
Requirement already up-to-date: keras-tuner in d:\anaconda3\lib\site-packages (1.1.2)
Requirement already satisfied, skipping upgrade: tensorboard in d:\anaconda3\lib\site-packages (from ke
Requirement already satisfied, skipping upgrade: numpy in d:\anaconda3\lib\site-packages (from keras-tu
Requirement already satisfied, skipping upgrade: ipython in d:\anaconda3\lib\site-packages (from keras-
Requirement already satisfied, skipping upgrade: requests in d:\anaconda3\lib\site-packages (from keras
Requirement already satisfied, skipping upgrade: kt-legacy in d:\anaconda3\lib\site-packages (from kera
Requirement already satisfied, skipping upgrade: packaging in d:\anaconda3\lib\site-packages (from kera
Requirement already satisfied, skipping upgrade: wheel>=0.26 in d:\anaconda3\lib\site-packages (from te
Requirement already satisfied, skipping upgrade: werkzeug>=1.0.1 in d:\anaconda3\lib\site-packages (fro
Requirement already satisfied, skipping upgrade: absl-py>=0.4 in d:\anaconda3\lib\site-packages (from t
Requirement already satisfied, skipping upgrade: google-auth<3,>=1.6.3 in d:\anaconda3\lib\site-package
(2.6.6)
Requirement already satisfied, skipping upgrade: markdown>=2.6.8 in d:\anaconda3\lib\site-packages (fro
Requirement already satisfied, skipping upgrade: google-auth-oauthlib<0.5,>=0.4.1 in d:\anaconda3\lib\s
as-tuner) (0.4.6)
Requirement already satisfied, skipping upgrade: setuptools>=41.0.0 in d:\anaconda3\lib\site-packages (
3.1.post20201107)
Requirement already satisfied, skipping upgrade: grpcio>=1.24.3 in d:\anaconda3\lib\site-packages (from
Requirement already satisfied, skipping upgrade: protobuf>=3.9.2 in d:\anaconda3\lib\site-packages (fro
4)
Requirement already satisfied, skipping upgrade: tensorboard-plugin-wit>=1.6.0 in d:\anaconda3\lib\site
tuner) (1.8.1)
Requirement already satisfied, skipping upgrade: tensorboard-data-server<0.7.0,>=0.6.0 in d:\anaconda3\
->keras-tuner) (0.6.1)
Requirement already satisfied, skipping upgrade: pickleshare in d:\anaconda3\lib\site-packages (from ip
Requirement already satisfied, skipping upgrade: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in d:\ana
hon->keras-tuner) (3.0.8)
Requirement already satisfied, skipping upgrade: traitlets>=4.2 in d:\anaconda3\lib\site-packages (from
Requirement already satisfied, skipping upgrade: jedi>=0.10 in d:\anaconda3\lib\site-packages (from ipy
Requirement already satisfied, skipping upgrade: pygments in d:\anaconda3\lib\site-packages (from ipyth
Requirement already satisfied, skipping upgrade: decorator in d:\anaconda3\lib\site-packages (from ipyt
Requirement already satisfied, skipping upgrade: backcall in d:\anaconda3\lib\site-packages (from ipyth
Requirement already satisfied, skipping upgrade: colorama; sys_platform == "win32" in d:\anaconda3\lib\
tuner) (0.4.4)
Requirement already satisfied, skipping upgrade: idna<4,>=2.5; python_version >= "3" in d:\anaconda3\li
as-tuner) (2.10)
Requirement already satisfied, skipping upgrade: urllib3<1.27,>=1.21.1 in d:\anaconda3\lib\site-package
5.11)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in d:\anaconda3\lib\site-packages (
6.20)
Requirement already satisfied, skipping upgrade: charset-normalizer~=2.0.0; python_version >= "3" in d:
requests->keras-tuner) (2.0.12)
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.2 in d:\anaconda3\lib\site-packages (fr
Requirement already satisfied, skipping upgrade: six in d:\anaconda3\lib\site-packages (from packaging-
Requirement already satisfied, skipping upgrade: cachetools<6.0,>=2.0.0 in d:\anaconda3\lib\site-packag
sorboard->keras-tuner) (5.2.0)
Requirement already satisfied, skipping upgrade: rsa<5,>=3.1.4; python_version >= "3.6" in d:\anaconda3
h<3,>=1.6.3->tensorboard->keras-tuner) (4.8)
Requirement already satisfied, skipping upgrade: pyasn1-modules>=0.2.1 in d:\anaconda3\lib\site-package
orboard->keras-tuner) (0.2.8)
Requirement already satisfied, skipping upgrade: importlib-metadata>=4.4; python_version < "3.10" in d:
markdown>=2.6.8->tensorboard->keras-tuner) (4.11.4)
Requirement already satisfied, skipping upgrade: requests-oauthlib>=0.7.0 in d:\anaconda3\lib\site-pack
```

```
5,>=0.4.1->tensorboard->keras-tuner) (1.3.1)
Requirement already satisfied, skipping upgrade: wcwidth in d:\anaconda3\lib\site-packages (from prompt
0.0->ipython->keras-tuner) (0.2.5)
Requirement already satisfied, skipping upgrade: ipython-genutils in d:\anaconda3\lib\site-packages (fr
uner) (0.2.0)
Requirement already satisfied, skipping upgrade: parso<0.8.0,>=0.7.0 in d:\anaconda3\lib\site-packages
ner) (0.7.0)
Requirement already satisfied, skipping upgrade: pyasn1>=0.1.3 in d:\anaconda3\lib\site-packages (from
"3.6"->google-auth<3,>=1.6.3->tensorboard->keras-tuner) (0.4.8)
Requirement already satisfied, skipping upgrade: zipp>=0.5 in d:\anaconda3\lib\site-packages (from impo
< "3.10"->markdown>=2.6.8->tensorboard->keras-tuner) (3.4.0)
Requirement already satisfied, skipping upgrade: oauthlib>=3.0.0 in d:\anaconda3\lib\site-packages (fro
auth-oauthlib<0.5,>=0.4.1->tensorboard->keras-tuner) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
```

- # Costruzione del modello con Iperparametri
  - La costruzione di un modello di rete neurale è simile a quelle viste precede
  - Si decide quali parametri si vogliano cercare o quali strati aggiungere al m
  - Tramite l'utilizzo della variabile `hp` è possibile definire valori di diverso tipo
    e `Choice` , a seconda di ciò che si vuole ottimizzare.
  - Ad ogni `trial` verrà creato un nuovo modello, con valori tra l'intervallo sel
  - Alla fine della ricerca sarà possibile visualizzare gli strati e i parametri di tu

In [76]:
```
from keras_tuner.tuners import RandomSearch
from keras_tuner.engine.hyperparameters import HyperParameters

def build_model(hp):
    model = Sequential()
    model.add(Conv2D(filters=64, kernel_size=(5, 5), padding='Same', activation="
    model.add(BatchNormalization())
    model.add(Conv2D(filters=64, kernel_size=(5, 5), padding='Same', activation="
    model.add(BatchNormalization())

    for e in range(hp.Int("n_sequence", 1, 2)):
        for i in range(hp.Int("n_layers", 1, 2)):
            model.add(Conv2D(
                filters=hp.Int(f"conv_[{e},{i}]", min_value=64, max_value=128, st
                kernel_size=(3, 3),
                padding='Same',
                activation="relu"))
            model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)) if e == 1 else MaxPooling2D(pool_
        model.add(Dropout(hp.Float(f"dropout_{e}", min_value=0.1, max_value=0.25,

    model.add(Flatten())
    model.add(Dense(hp.Int(f"dense", min_value=64, max_value=256, step=64), activ
    model.add(BatchNormalization())
    model.add(Dropout(hp.Float("dropout_out", min_value=0.1, max_value=0.25, step
    model.add(Dense(10, activation="softmax"))
    model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metri

    return model
```

- # Ricerca del modello migliore
  - Una volta specificato il tipo di tuner (nel mio caso **RandomSearch**), è nece
    - la funzione che restituirà il modello;
    - Il numero di differenti modelli da testare (`max_trials`);
    - Il numero di esecuzione dello stesso modello ad ogni passo;

- la metrica dell'errore da ottimizzare (in questo caso `val_accuracy` , con
  ora);
- la cartella in cui saranno salvati tutti i modelli eseguiti.

In [77]:
```
tuner = RandomSearch(
    build_model,
    objective = "val_accuracy",
    seed=42,
    max_trials = 10,
    executions_per_trial = 1,
    directory = 'tuner_output',
    project_name="MNIST"
    )
```

- Per iniziare la ricerca è necessario, come per la funzione **fit** di keras, specificar
  - le variabili di training;
  - la `batch_size` , numero di input da analizzare prima di cambiare i pesi;
  - le `epochs` , numero di epoche di training;
  - l'insieme di validazione dei dati.

In [78]:
```
tuner.search(x=X_train,
             y=y_train,
             batch_size=128,
             epochs=8,
             validation_data=(X_val, y_val))
```

```
Trial 10 Complete [00h 01m 57s]
val_accuracy: 0.9941999912261963

Best val_accuracy So Far: 0.9947999715805054
Total elapsed time: 00h 16m 13s
INFO:tensorflow:Oracle triggered exit
```

- Per mostrare il risultato del miglior modello, **Keras Tuner** offre la lista di tutti i m
  sessione.

In [79]:
```
tuner.get_best_models()[0].evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0152 - accuracy: 0.9948

[0.015177389606833458, 0.9947999715805054]
```

Conclusioni

- Utilizzando reti neurale di tipo convoluzionale è possibile classificare con un'ac
l'insieme di immagini fornite nel MNIST.
- Servendomi di dati del 2018, trovati su Kaggle, il modello di CNN da me propos
top 20-15% dei punteggi ottenuti.
- Un livello di accuratezza migliore si potrebbe ottenere con dataset specifici, che
rispetto a quello proposto dal MNIST.
- Si potrebbe ancora migliorare il modello testando ulteriormente altri iperparame
Tuner.

- Una volta ottenuta un'ottima configurazione, è possibile salavare, tramite l'utiliz
`tuner` , contenente tutti i modelli testati.

In [80]:
```
import pickle
```

In [81]:
```
with open("tuner.bin", "wb") as f:
    pickle.dump(tuner, f);
```

- Per poter utilizzare un `tuner` salvato precedentemente, è necessario utilizzare

In [82]:
```
with open("tuner.bin", "rb") as f:
    load_tuner = pickle.load(f)
```

- È possibile analizzare e valutare singolarmente ogni modello, contenuto nella s
eseguita.

In [83]:
```
model = load_tuner.get_best_models()[0];
```

```
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is being deleted with unres
s for the specific values in question. To silence these warnings, use `status.expect_partial()`. See htt
rain/Checkpoint#restorefor (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#restorefor) details about the st
function.
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer.iter
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer.beta
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer.beta
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer.deca
WARNING:tensorflow:Value in checkpoint could not be found in the restored object: (root).optimizer.lear
```

In [84]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 64)        1664

 batch_normalization (BatchN  (None, 28, 28, 64)       256
 ormalization)

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        102464

 batch_normalization_1 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        36928

 batch_normalization_2 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 14, 14, 64)       0
 )

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_3 (Conv2D)           (None, 14, 14, 96)        55392

 batch_normalization_3 (Batc  (None, 14, 14, 96)       384
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 96)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 96)          0

 flatten (Flatten)           (None, 4704)              0

 dense (Dense)               (None, 256)               1204480

 batch_normalization_4 (Batc  (None, 256)              1024
 hNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,405,674
Trainable params: 1,404,586
Non-trainable params: 1,088
_________________________________________________________________
```

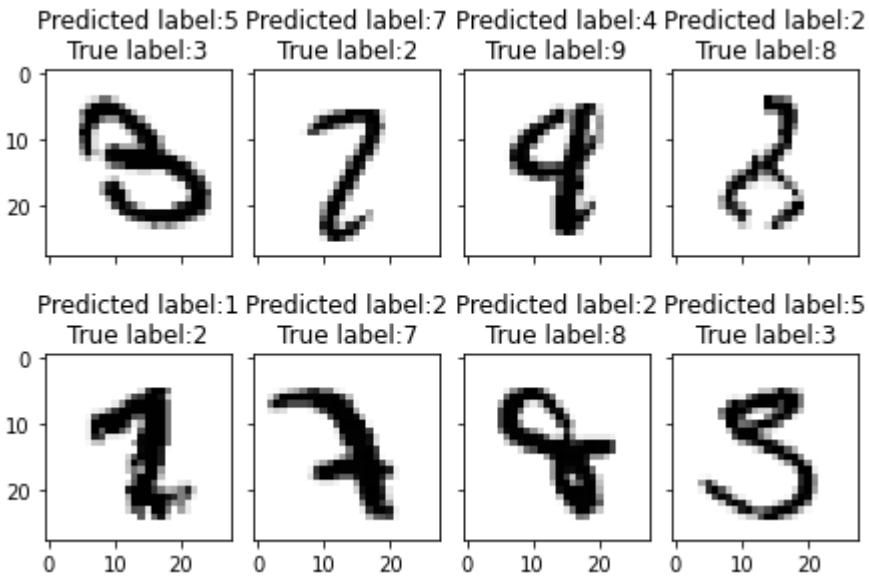
In [85]:
```
model.evaluate(X_val, y_val)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0152 - accuracy: 0.9948

[0.015177389606833458, 0.9947999715805054]
```

In [86]:
```
y_pred = model.predict(X_val)
```

```
313/313 [==============================] - 1s 2ms/step
```

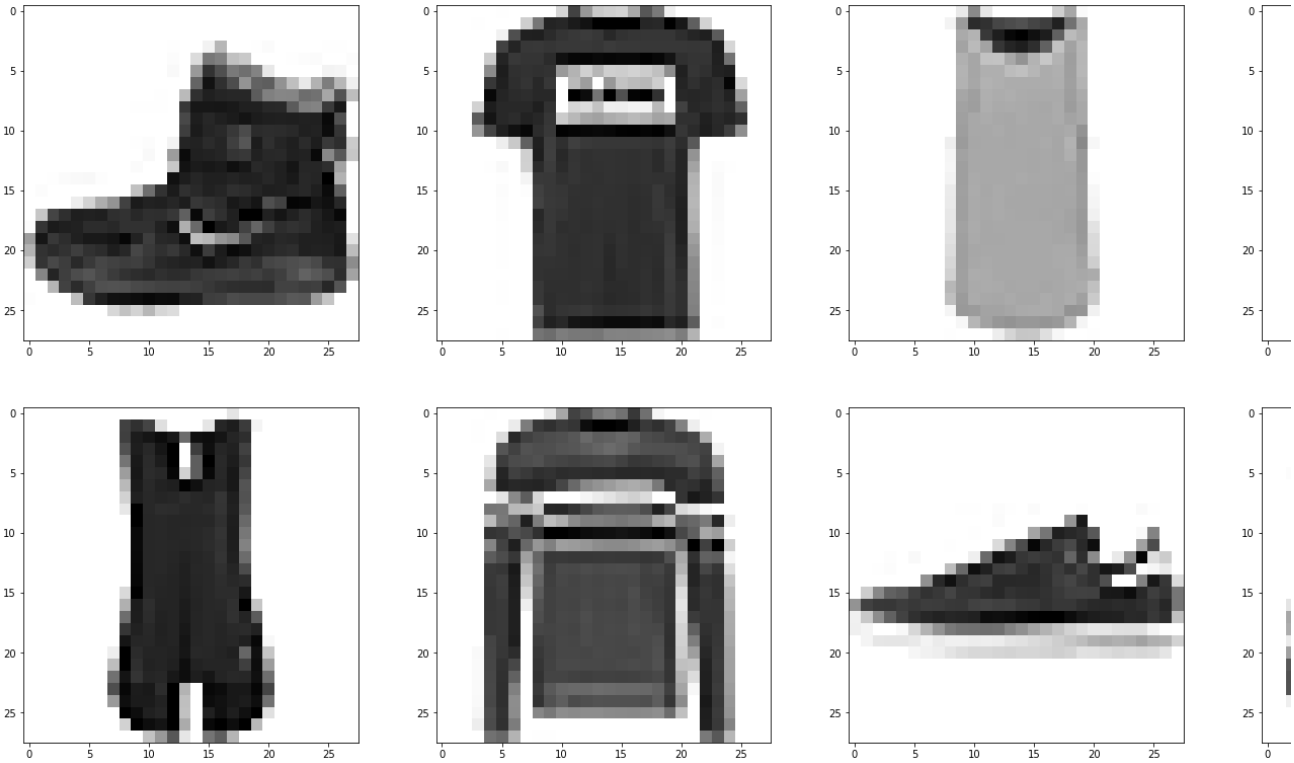
In [87]:
```
display_errors(y_val, y_pred, 8)
plt.show();
```



- # Fashion MNIST
  - Per interesse personale ho provato a testare la stessa architettura di rete r che offre il MNIST.
  - Il Fashion MNIST propone un dataset di indumenti o accessori di moda.
  - In questo caso ho utilizzato, per comodità, il dataset che mette a disposizio

In [88]:
```
fashion_data = tf.keras.datasets.fashion_mnist

(fashion_X_train, fashion_y_train), (fashion_X_val, fashion_y_val) = fashion_data
```

In [89]:
```
plt.figure(figsize=(28, 28))
# mostriamo sono i primi 8 numeri presenti nel dataset
for x in range(8):
    # dimensione massima del subplot 4x4 (16 numeri rappresentabili al massimo)
    plt.subplot(4, 4, x+1)
    plt.imshow(fashion_X_train[x], cmap=plt.cm.binary);
plt.show()
```



- Applichiamo le stesse operazioni di normalizzazione dei dati.

In [90]:
```
fashion_X_train = fashion_X_train / 255
fashion_X_val = fashion_X_val / 255
```

- Cloniamo la struttura della migliore rete convoluzionale proposta da **Keras Tun**

In [91]:
```
fashion_model = tf.keras.models.clone_model(model)
```

In [92]:
```
fashion_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", m
```

In [93]:
```
fashion_model.fit(fashion_X_train, fashion_y_train, batch_size=128, epochs=8, val
```

```
Epoch 1/8
469/469 [==============================] - 10s 21ms/step - loss: 0.3812 - accuracy: 0.8627 - val_loss: 
Epoch 2/8
469/469 [==============================] - 11s 24ms/step - loss: 0.2470 - accuracy: 0.9096 - val_loss: 
Epoch 3/8
469/469 [==============================] - 10s 21ms/step - loss: 0.2124 - accuracy: 0.9215 - val_loss: 
Epoch 4/8
469/469 [==============================] - 9s 20ms/step - loss: 0.1829 - accuracy: 0.9334 - val_loss: 0
Epoch 5/8
469/469 [==============================] - 9s 20ms/step - loss: 0.1635 - accuracy: 0.9408 - val_loss: 0
Epoch 6/8
469/469 [==============================] - 9s 20ms/step - loss: 0.1458 - accuracy: 0.9450 - val_loss: 0
Epoch 7/8
469/469 [==============================] - 10s 21ms/step - loss: 0.1319 - accuracy: 0.9513 - val_loss: 
Epoch 8/8
469/469 [==============================] - 10s 21ms/step - loss: 0.1270 - accuracy: 0.9526 - val_loss: 
```

```
<keras.callbacks.History at 0x2893751f910>
```

- Anche con questo set di dati, otteniamo un ottimo risultato, tra il 15-10% delle r