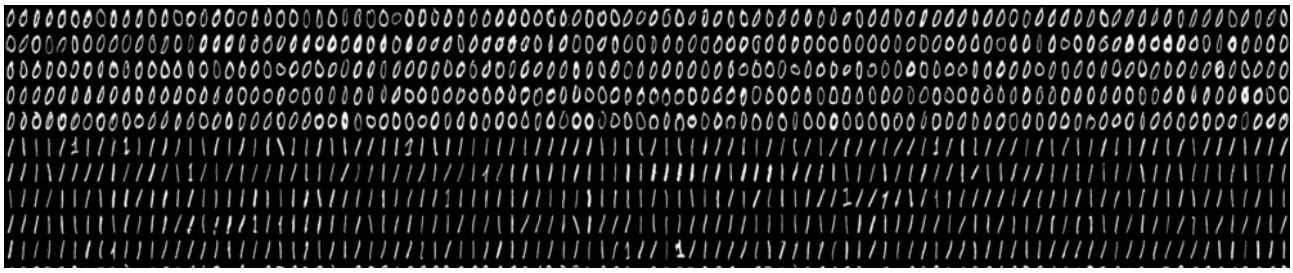
Homework 4 Machine Learning



In questo Homework è stato implementato un sistema di apprendimento e riconoscimento automatico di cifre scritte a mano. Il sistema, per l'apprendimento utilizza il noto DataSet **MNIST** (National Institute of Standards Technology) questo set è costituito da cifre scritte a mano da 250 persone differenti.

Classi	Descrizione
NeuralNetwork_SVM	Dato i training l'algoritmo genera un iperpiano che classifica i nuovi esempi. Cerca l'iperpiano con distanza minima maggiore rispetto agli esempi di addestramento così da massimizzare il margine dei dati di allenamento **Metodo** — **Descrizione** Train() — Apprendimento Predict() — Predizione
NeuralNetwork_KNearest	Classifica gli oggetti basandosi sulle caratteristiche degli oggetti vicini a quello analizzato, assegnandolo ad una classe se questa è la più frequente fra i K esempi vicini a lui. **Metodo** — **Descrizione** Train() — Apprendimento Predict() — Predizione
MLHW4	**Metodo** — **Descrizione** split() — Questo metodo suddivide l'input training image in piccole celle di una singola cifra. loadTraining() — Carica il file di training getdigits() — Restituisce le singole cifre estratte createOutPutFile() — Crea i file di output con il riconoscimento start() — Inizializza lo stato iniziale per l'esecuzione

Librerie usate

- numpy
- scipy.misc.pilutil
- cv2
- skimage.feature
- matplotlib
- sklearn.model_selection
- sklearn.metrics
- sklearn.utils

Esecuzione del progetto

Il progetto è sviluppato su due file, il primo file (**MLHW4.py**) è composto dalle classi che si è già spiegato in precedenza, il secondo file (**main.py**) contiene semplicemente la chiamata al metodo **Start** della classe **MLHW4.** L'esecuzione del file avviene chiamando quest'ultimo e passandogli il file di training e il file di test.

```
$ python main.py Training.png Test-M.png
```

L'esecuzione del progetto si suddivide in **4 STEP** principali, nel primo passo viene importato il TrainingSet, nel secondo passo viene lanciato l'apprendimento **K-Nearest** che attribuisce all'oggetto analizzato la classe più frequente tra i K esempi più vicini, nel terzo passo viene lanciato il training **SVM** (**Support Vector Machine**) che adotta una politica differente da quella del **K-Nearset**, in quanto quest'ultimo cerca l'iperpiano con distanza minima maggiore rispetto agli esempi di addestramento, così da massimizzare il margine dei dati di allenamento. Per concludere vengono creati i file di output con il risultato della valutazione eseguita dalla macchina, di seguito vediamo l'esempio di output dato dalla shell:

```
 1) Loading "Training.png for training" ...
('Train: ', (5000, 28, 28))
('Test: ', (5000,))

 2) Training K-Nearest Neighbor
('Accuracy: ', 0.9145454545454546)

 3) Training Support Vector Machine
('Accuracy: ', 0.9678787878787879)

 4) Generate Files
casa@casa-KX633AA-ABZ-a6585-it:~/Desktop/MLHW4.marco.dinatale$
```

Ora vediamo il risultato dell'apprendimento della macchina. Passando in input un'immagine con delle cifre scritte a mano il programma è in grado di riconoscere i caratteri numerici associandogli la loro rispettiva cifra digitale. Vediamo l'esempio di output prodotto:

FILE DI TEST

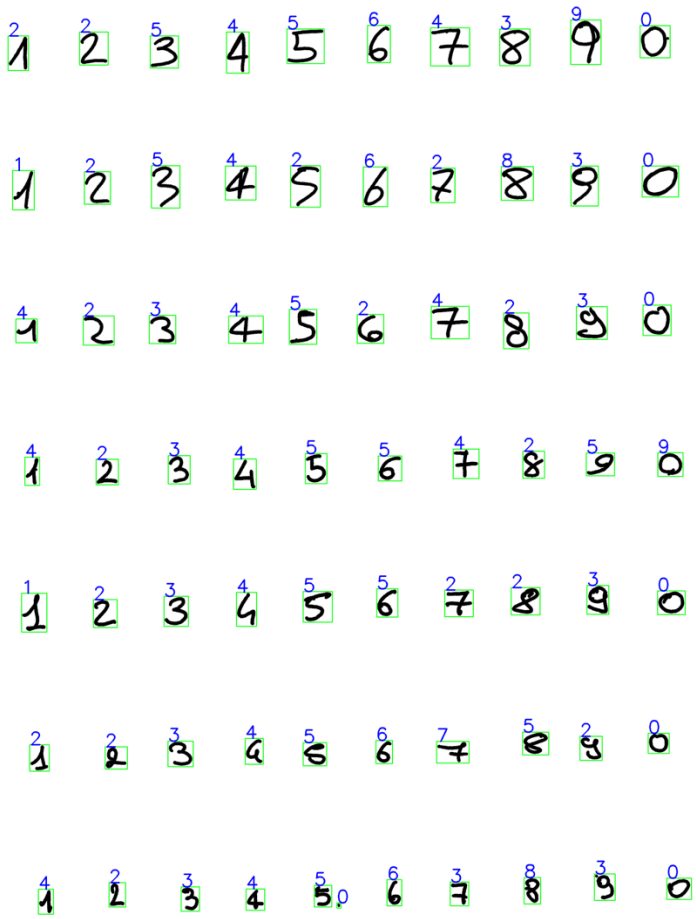
FILE DI COMPARAZIONE

FILE DI SOSTITUZIONE

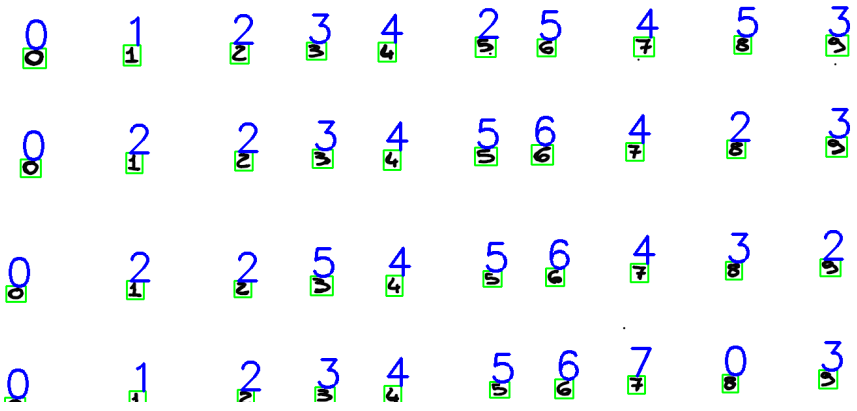
Gli output prodotti dal sistema sono auto generati nella cartella output, il file di test utilizzato per questo esempio (**Test-M.png**) è riportato nella folder **test**.

Test Effettuati

1) Nell'esempio riportato è stato preso un campione di 7 persone e ad ognuna di questa è stato chiesto di scrivere le cifre da 0 a 9 su di una riga, vediamo più in dettaglio il comportamento ottenuto dalla valutazione di questo test:



2) Nell'esempio che segue, ho scritto su 4 righe i numeri da 0 e 9, l'outupt prodotto da sistema è riportato di seguito:

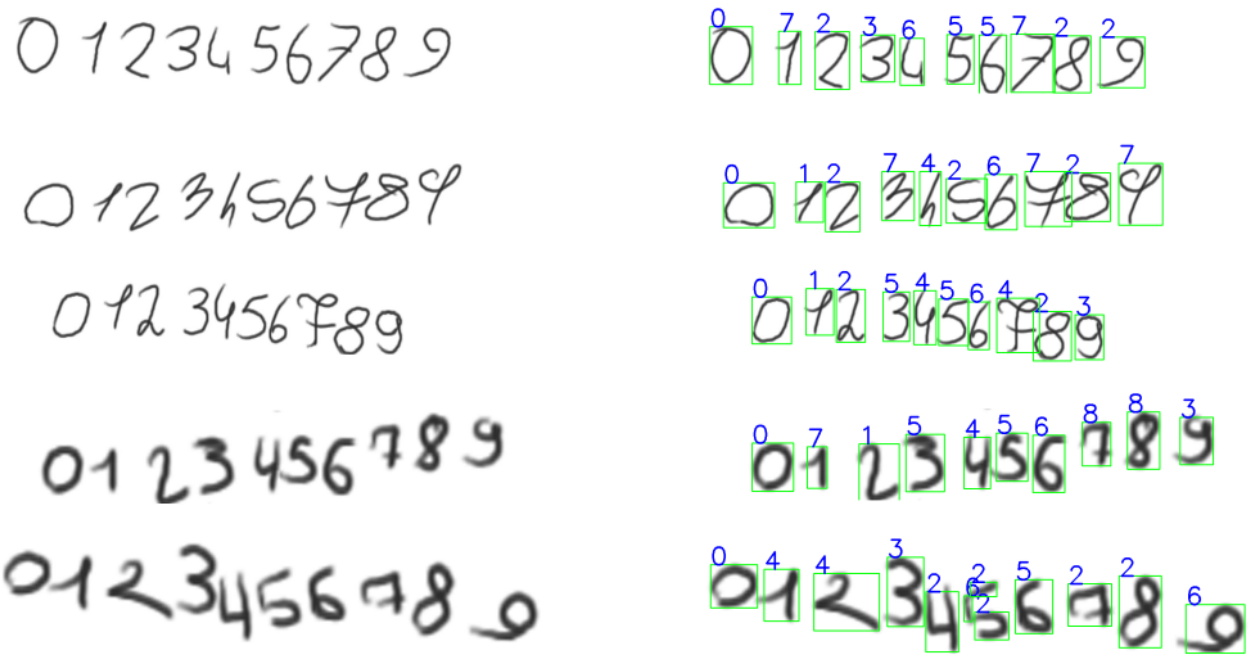


Per quanto riguarda la percentuale di precisione si è deciso di far testare il progetto ad un campione di 30 persone di cui:

- 5 con un'età compresa tra i 6 e i 10 anni,
- 5 con un'età compresa tra gli 11 ed i 13 anni,
- 5 con un'età compresa tra i 14 e i 20 anni,
- 10 con un'età compresa tra i 21 e i 59 anni,
- 5 con un'età compresa tra i 60 e i 80 anni.

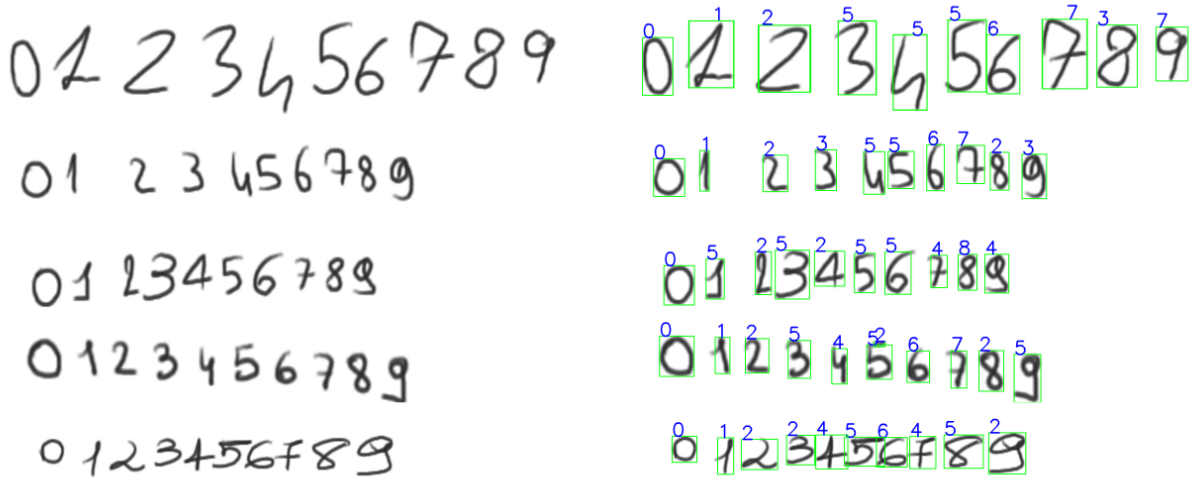
Si è scelto di suddividere le persone per età poiché abbiamo visto che, a seconda dell'età del tester, la percentuale di precisione cambia: infatti un bambino delle elementari avrà una grafia molto differente da quelle campionate nel dataset, dato che il campione comprende 250 grafie di studenti di una scuola superiore americana, e questo vale per tutte le età. Quindi non incide solo la grafia sulla percentuale di precisione, ma anche l'età del tester. I nostri test hanno riportato i seguenti risultati:

Tester di età compresa tra i 6-10 anni



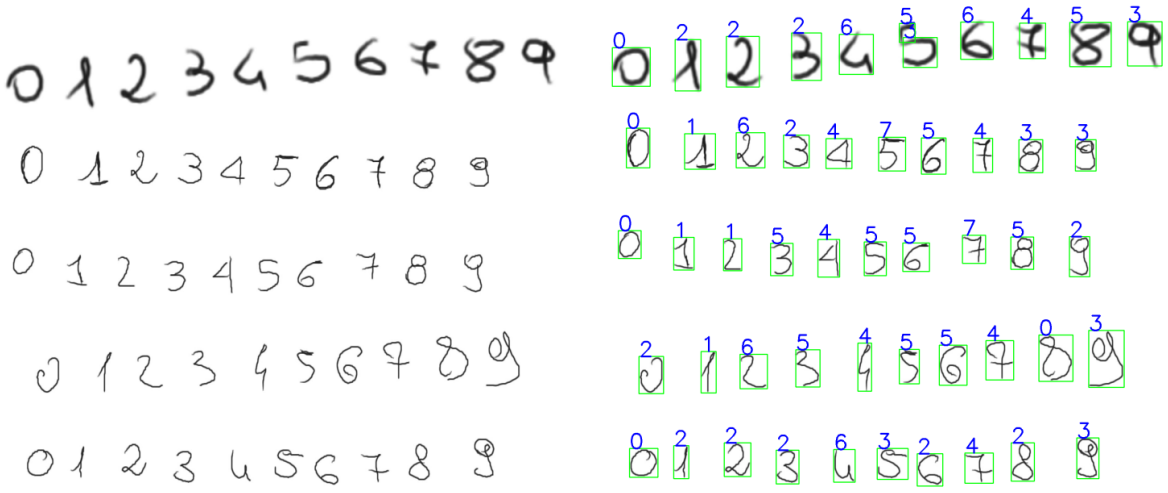
Percentuale di errore: **4,8%**

Tester di età compresa tra i 11-13 anni



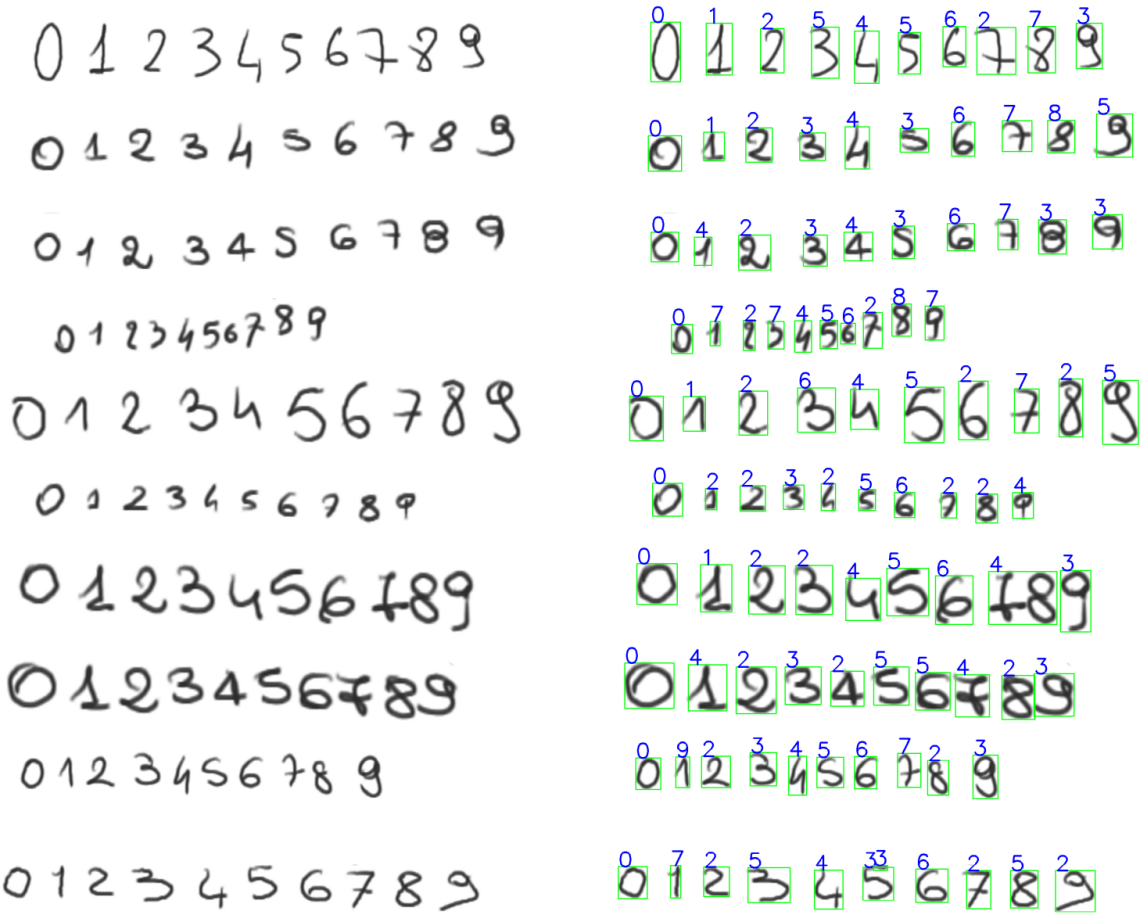
Percentuale di errore: **3,8%**

Tester di età compresa tra i 14-20 anni



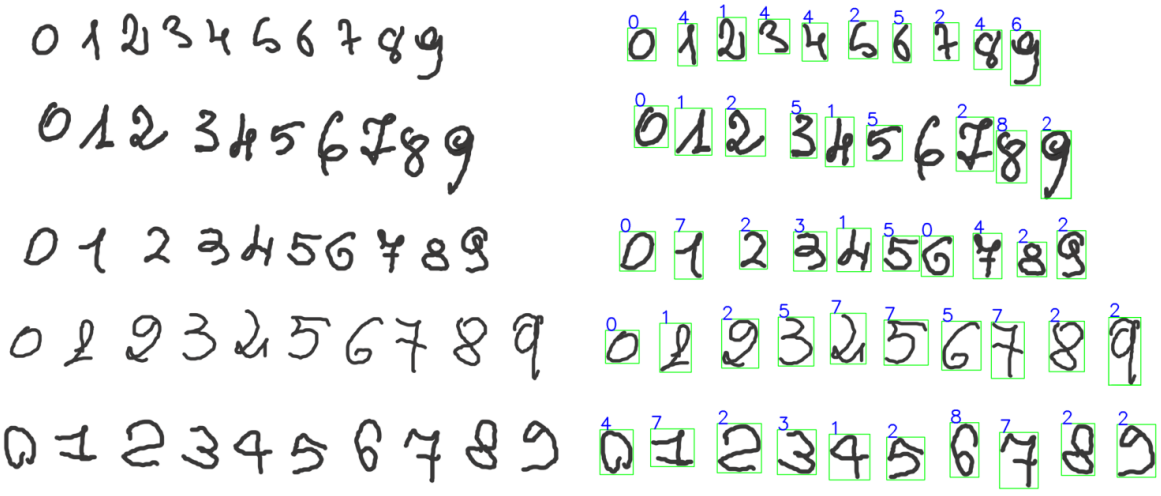
Percentuale di errore: **6,6%**

Tester di età compresa tra i 21-59 anni



Percentuale di errore: **4.7%**

Tester di età compresa tra i 60-80 anni



Percentuale di errore: **6,4%**

Rapporto età errore

Riferimenti

Per comprendere meglio l'algoritmo e per scrivere una corretta implementazione del sistema realizzato, si è fatto uso della seguente documentazione

1) Machine Learning PYTHON – Costruire algoritmi per generare conoscenza
2) URL (SVM in OpenCV) - https://docs.opencv.org/trunk/d1/d73/tutorial_introduction_to_svm.html
3) URL (evitare i loop di cifre nelle cifre) – https://docs.opencv.org/trunk/d9/d8b/tutorial_py_contours_hierarcy.html

L'elaborato è stato sviluppato in collaborazione con Ciavarro Cristina (253188)

Di Natale Marco	Ciavarro Cristina
CLASS:NeuralNetwork_SVM CLASS:MLHW4	CLASS:NeuralNetwork_KNearest CLASS:MLHW4