

Progetto Sistemi Intelligenti - Alberto Borghese

Riconoscitore di cifre manoscritte

Francesco Bertolotti

August 7, 2018

1 Introduzione

Come concordato, ho deciso di sfruttare alcuni concetti mostrati durante il corso, per implementare un riconoscitore di caratteri manoscritti, sfruttando le reti neurali. Il progetto è stato svolto in python utilizzando diverse librerie. E' stato utilizzato il dataset del MNIST.

2 Dipendenze

Le librerie utilizzate in python sono elencate di seguito:

1. Keras[4]: Una nota API ad alto livello per reti neurali.
2. numpy: Un pacchetto per il calcolo scientifico.
3. optparse: Un pacchetto per la gestione di opzioni CLI.
4. tensorflow[5]: engine backend per keras, capace di sfruttare anche le GPU.
5. opencv[6]: API per la gestione di immagini.

3 Modelli

Per costruire un riconoscitore efficace sono stati testati diversi modelli:

1. MLPNN: Una rete a diversi livelli completamente connessa.
2. CNN[3]: Una rete convoluzionale che sfrutta kernel di dimensioni ridotte per la classificazione delle immagini.
3. Committees: Comitati di esperti che prendono decisioni basate sulla maggioranza.

Di seguito il modello di rete convoluzionale utilizzato per il riconoscitore.

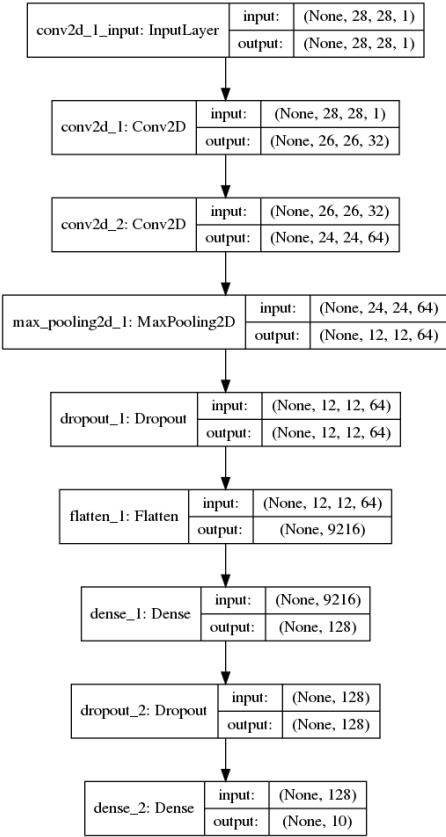


Figure 1: Struttura della rete convoluzionale

Vengono utilizzati 32 e successivamente 64 filtri convoluzionali di dimensione 3x3. A questo punto viene applicato un filtro di max pooling per ridurre ulteriormente la dimensionalità (9216 output), in ultimo applichiamo un MLPNN completamente connesso, con un singolo hidden layer di 128 neuroni, per distinguere le classi. Inoltre è stato utilizzato un dropout di 0.2 sui singoli pesi.

Mostriamo la figura relativa a uno dei modelli MLPNN utilizzati (ne sono stati provati diversi).

Come funzione di attivazione è sempre stata utilizzata la RELU per i layer nascosti e la SoftMax per i layer di output, per avere una interpretazione probabilistica sugli output della

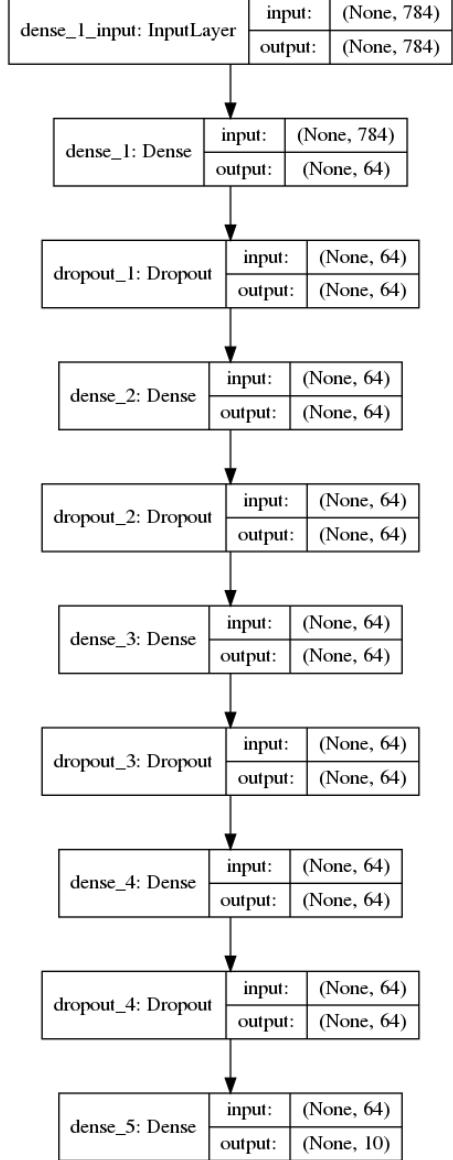


Figure 2: Struttura della rete neurale

In questo caso, ho utilizzato 5 layer ciascuno da 64 neuroni più l'ultimo da 10. Anche questa volta con un dropout di 0.2.

Per entrambi i modelli l'output è rappresentato da un vettore di 10 elementi di valore compreso tra 0 e 1, di cui solo il massimo è rappresentante della classe associata.

rete.

4 Trainig and Testing

Il dataset utilizzato è quello classico del mnist e anche come partizionamento tra train e test set è stato utilizzato quello proposto dal dataset originale.

Per quanto riguarda l'addestramento è stato utilizzato il classico algoritmo di discesa del gradiente [1], in particolare è stato utilizzata una variante detta rmsprop[2] che va a mediare il nuovo aggiornamento (dei pesi) con quelli passati, in modo tale da avere una discesa più regolare.

Come Loss function ad ogni iterazione si ha 0 se la classe proposta dalla rete è corretta 1 altrimenti.

La dimensione dei batch utilizzata è solitamente 128, ma mi sono preso la libertà di provare con dimensioni differenti: 1, 128, 512, 1024 e full.

Lo script trainer.py si occupa per l'appunto di effettuare l'addestramento della rete neurale, e al lancio di quest'ultimo è possibile inserire diverse opzioni da linea di comando.

1. ”-d”, ”–directory”: permette di scegliere una cartella di destinazione per il salvataggio del modello.
2. ”-s”, ”–batch”: permette di scegliere la dimensione del batch con cui effettuare l'addestramento.
3. ”-e”, ”–epochs”: permette di scegliere il numero di epoche per cui si vuole fare l'addestramento.
4. ”-b”, ”–balance”: permette di specificare se si vuole effettuare un bilanciamento delle diversi classi prima di cominciare l'addestramento
5. ”-c”, ”–committee”: permette di selezionare il numero di reti che si vogliono addestrare, con lo scopo di generare una pool di esperti che prenderà le decisioni per maggioranza.
6. ”-m”, ”–modelname”: permette di scegliere il tipo del modello, questi sono descritti nel file ”models.py”

Ad ogni addestramento verrano generati i seguenti file:

1. ”test_loss_acc.png”, ”train_loss_acc.png”: sono il plot dell'accuratezza e della loss rispetto al training set e al test set generato epoca per epoca.
2. ”model.png”: l'immagine rappresentante il modello.
3. ”model0.best.hdf5”: il migliore modello addestrato in termini di accuratezza e loss.

Di seguito l'andamento di loss e accuracy per quanto riguarda la reti:

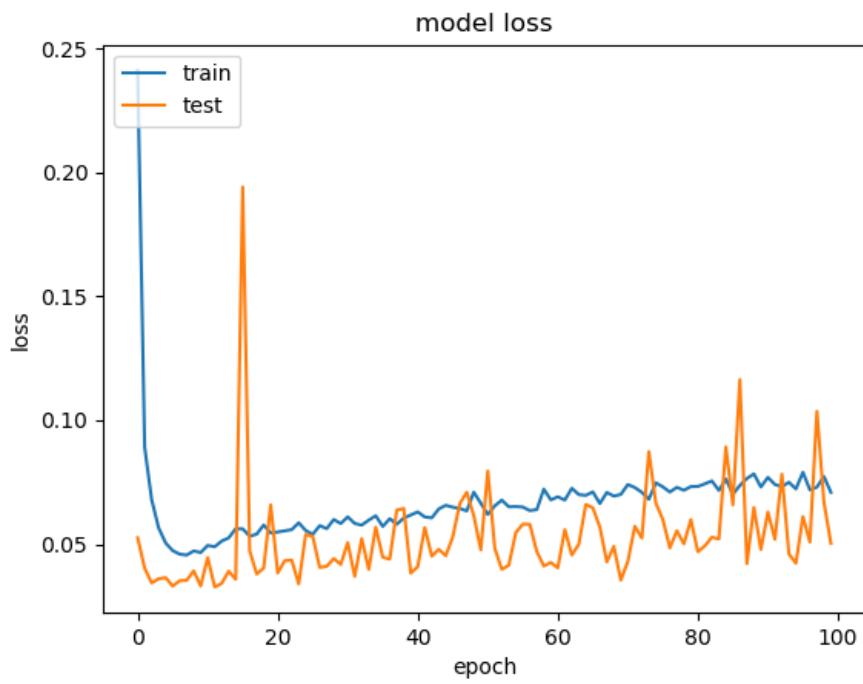


Figure 3: loss della rete convoluzionale

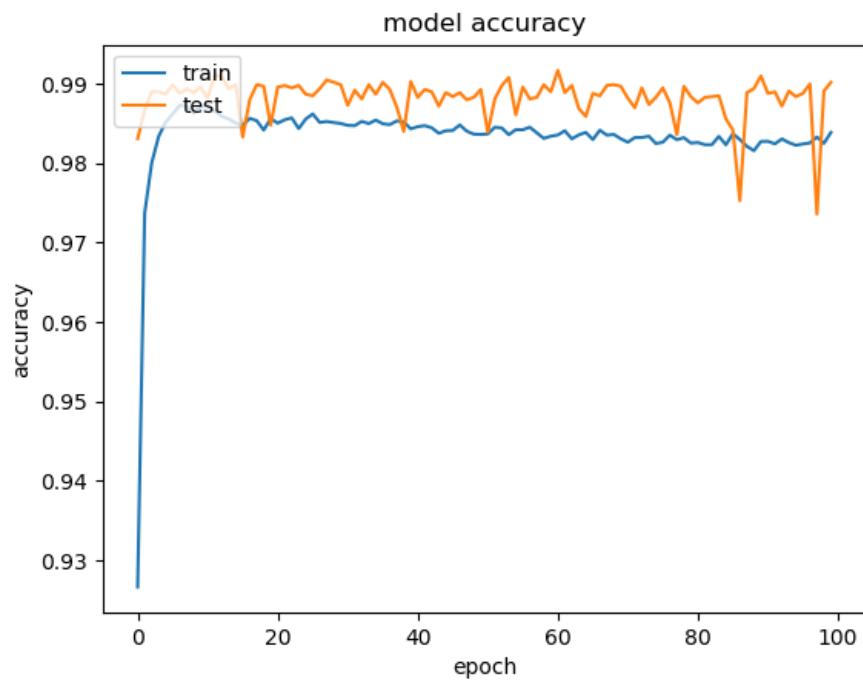


Figure 4: accuracy della rete convoluzionale

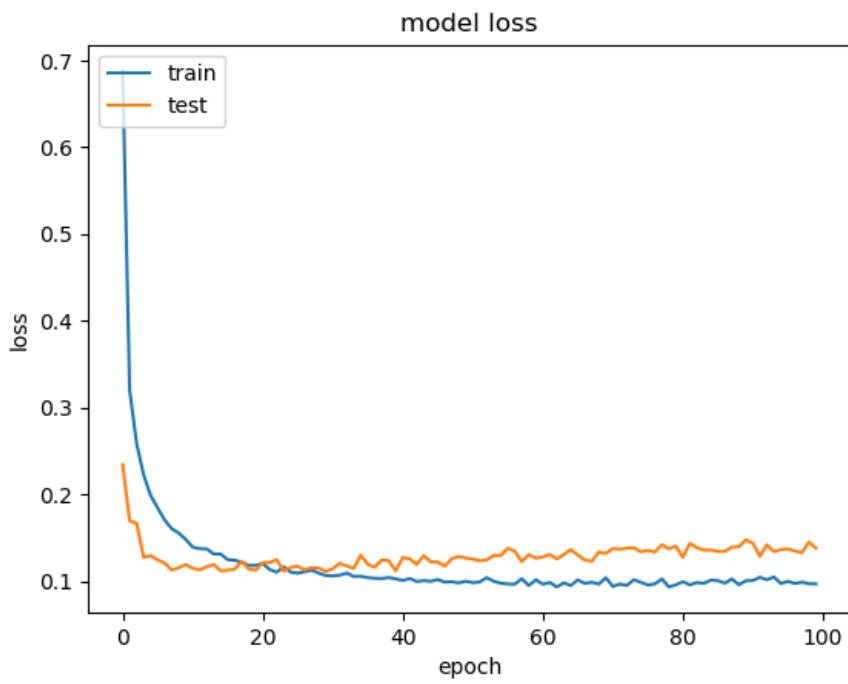


Figure 5: loss del percetrono multilivello

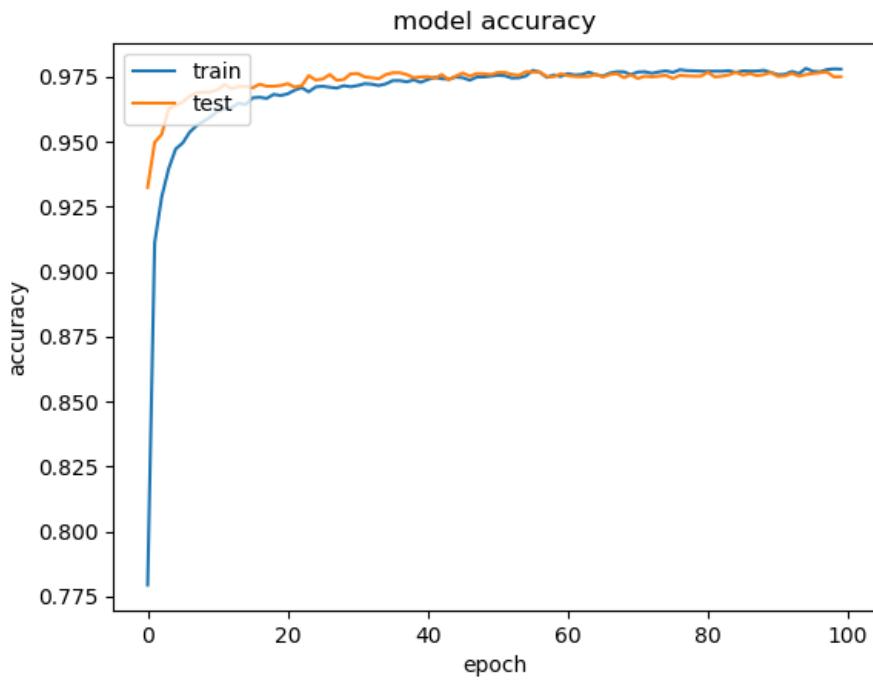
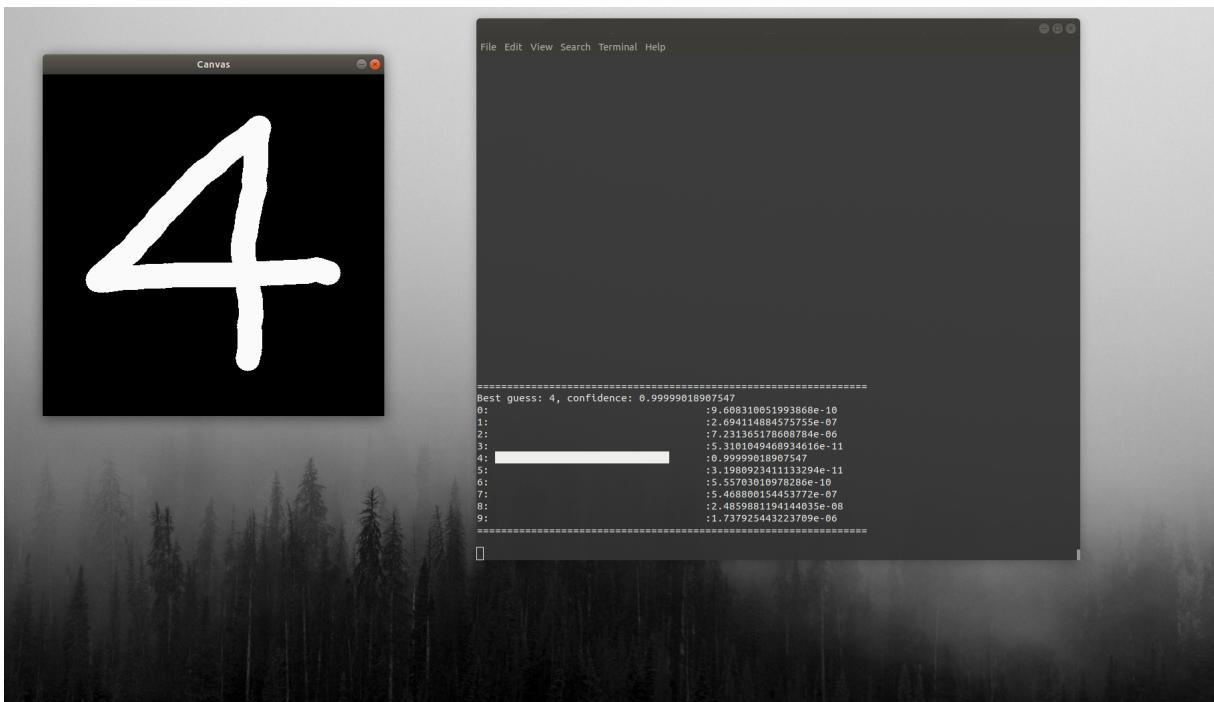
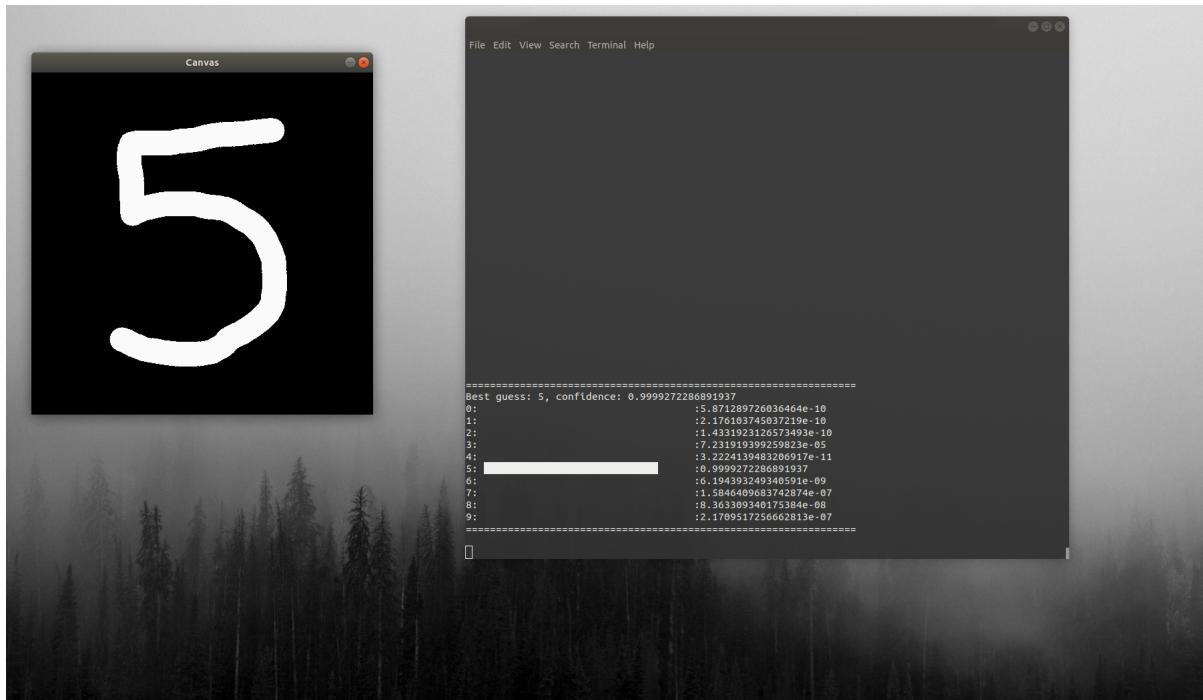
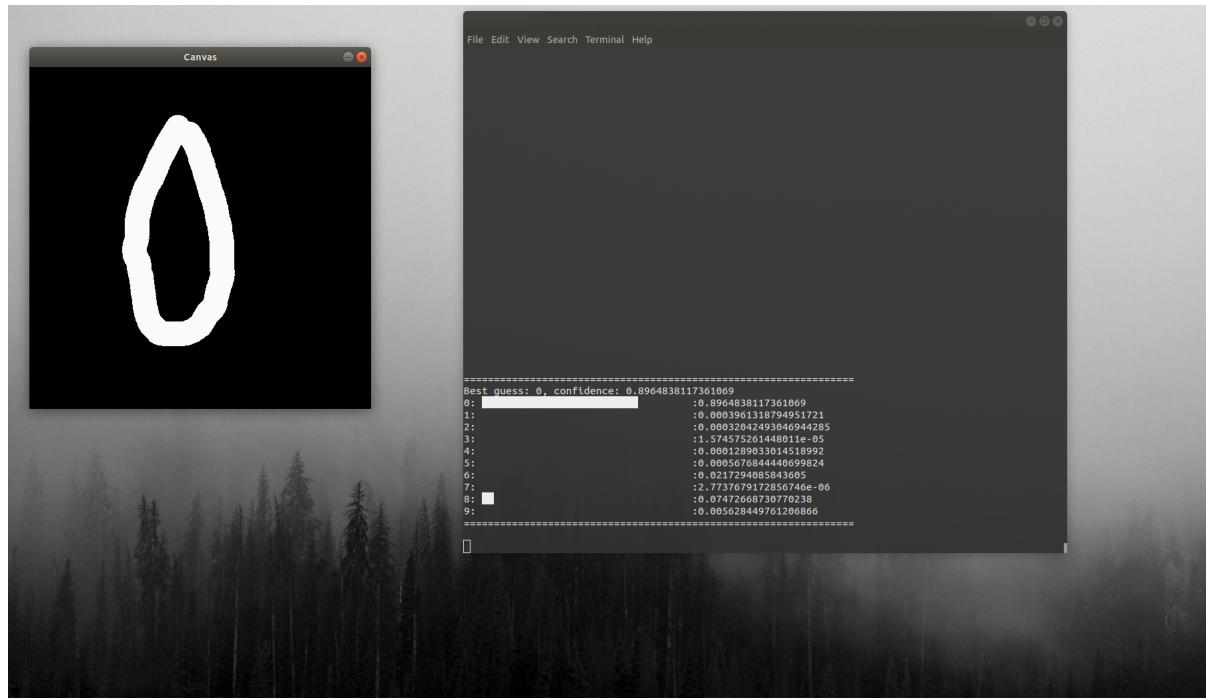
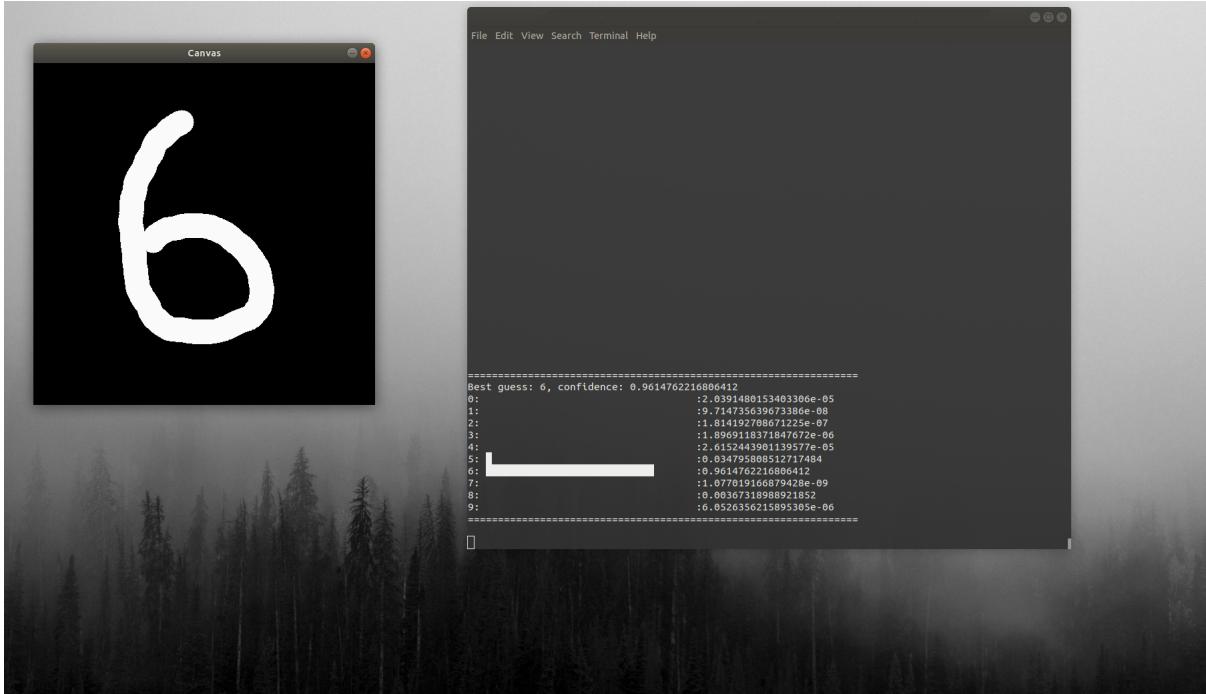


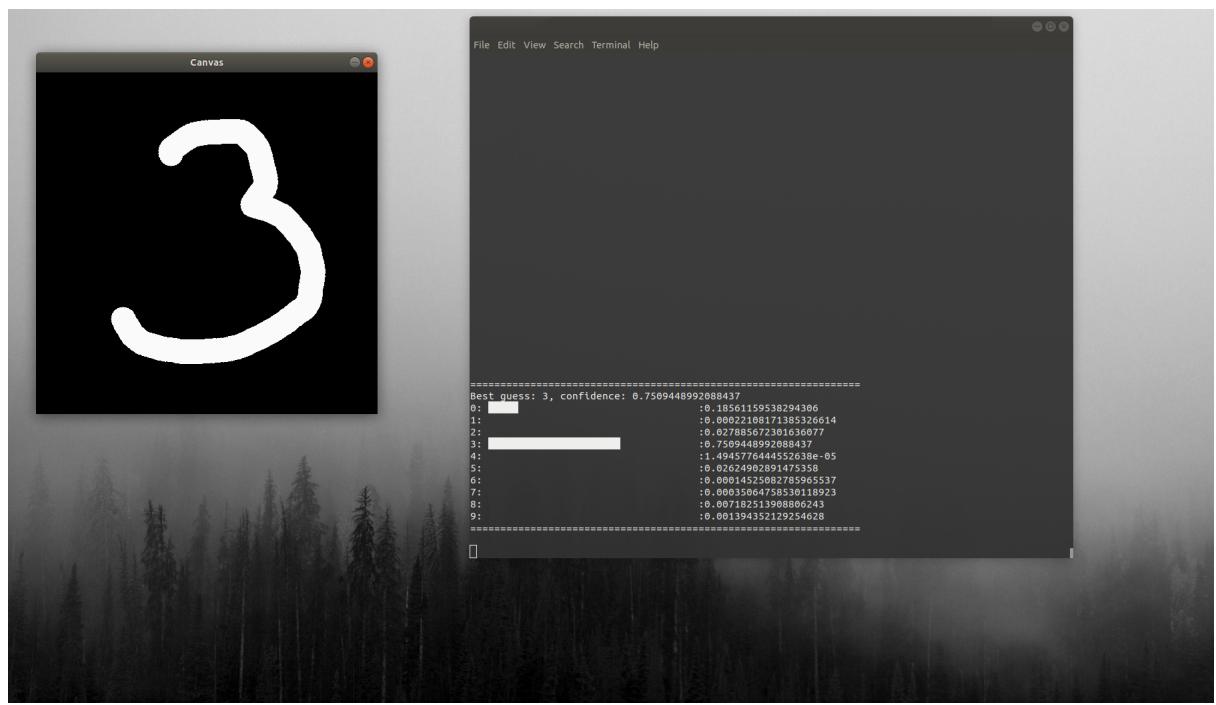
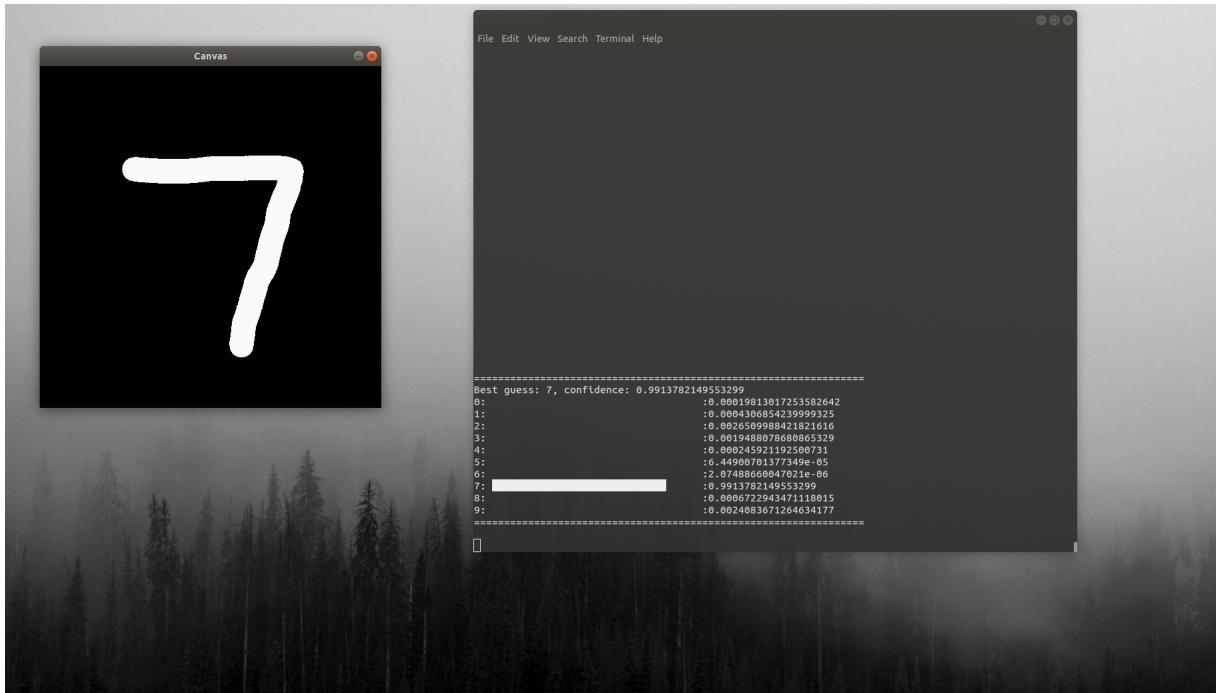
Figure 6: accuracy del percetrono multilivello

5 Using

Una volta addestrato il modello viene utilizzato dallo script recognizer.py che carica inizialmente uno o più reti e poi le sfrutta per riconoscere le cifre disegnate su di una finestra aperta tramite opencv[6]. Alcuni esempi sono dati nelle immagini seguenti:







In questo caso il programma utilizzava un comitato di 20 reti convoluzionali per prendere le decisioni.

6 References

References

- [1] Claude Lemarcéchal,
https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf, 2010.
- [2] Geoffrey E. Hinton,
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
- [3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [4] <https://github.com/keras-team/keras>
- [5] <https://www.tensorflow.org/>
- [6] <https://opencv.org/>