Intelligenza Artificiale and Data Analytics

INTRODUZIONE AL MACHINE LEARNING

Report challenge three

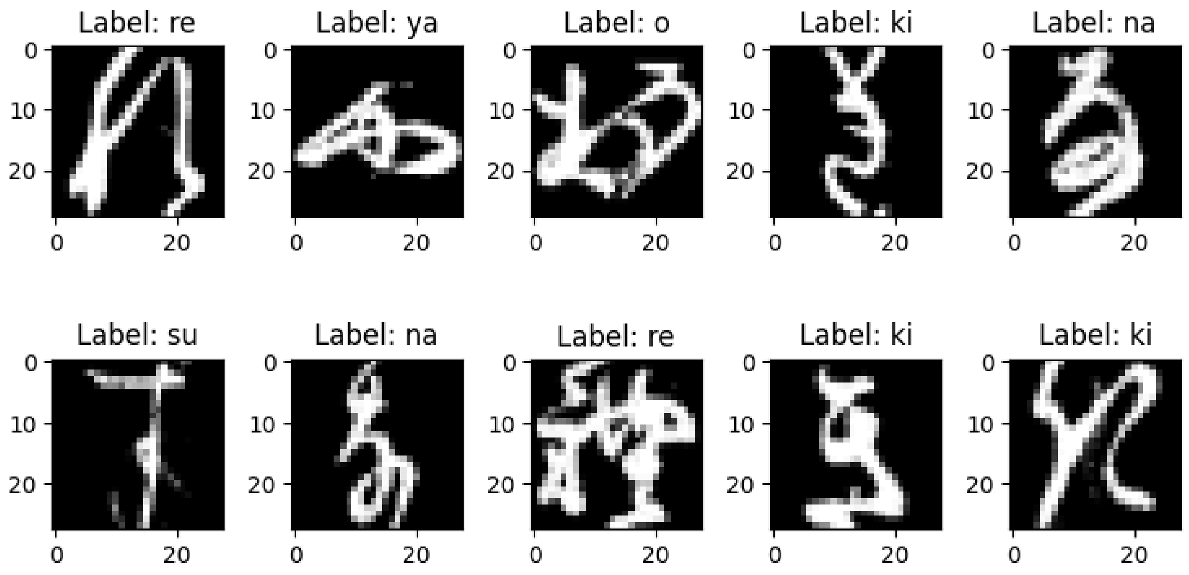
Alberto Facchin

Maggio 2024

Gruppo: Facchin Alberto, Micheluz Leonardo, Tonello Andrea

1. Data exploration

Starting from the dataset, it’s name is KMNIST and, like MNIST, it contains handwritten symbols, in the case of KMNIST they are Japanese hiragana symbols.
The set has 70k observations, 60k for the training and 10k for the testing.



The 28x28 pixel images are labeled with 10 classes, which are: ['o', 'ki', 'su', 'tsu', 'na', 'ha', 'ma', 'ya', 're', 'wo'].

2. Model building

For what the model building concerns we chose to implement three models: a FCNN with 1 hidden layer, a CNN with 1 hidden layer and a CNN with 3 hidden layers.
We decided to try two CNNs with a different number of hidden layers in order to evaluate the trade-off between accuracy and number of layers, and also the differences, in terms of performances, between FCNNs and CNNs.
In every NN we used the ReLU (Rectified Linear Unit) activation function for the input and hidden layers, while the softmax for the logits.

FCNN with one hidden layer

- Input layer (linear): 28x28 → 32x26x26 (26 = image dimension - kernel size + 1)
- Hidden layer (linear): 32x26x26 → 128
- Output layer (linear): 128 → 10 (10 because it is the number of different classes)

CNN with one hidden layer

- Input layer (convolutional): 28x28 → 32x24x24 (24 = 28 - 5 + 1)

(max-pooling 2x2 → 24 / 2 = 12)
- Hidden layer (linear): 32x12x12 → 128
- Output layer (linear): 128 → 10 (10 because it is the number of different classes)

CNN with three hidden layers

- Input layer (convolutional): 28x28 → 32x24x24 (24 = 28 - 5 + 1)
  (max-pooling 2x2 → 24 / 2 = 12)
- Hidden layer (convolutional): 32x12x12 → 128x10x10 (10 = 12 - 3 + 1)
  (max-pooling 2x2 → 10 / 2 = 5)
- Hidden layer (linear): 128x5x5 → 256
- Hidden layer (linear): 256 → 64
- Output layer (linear): 64 → 10 (10 because it is the number of different classes)

3. Training

We trained our NNs with different learning rate values ([0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 3, 5]) and with two types of optimizer, Adam and SGD (Stochastic gradient descent). For all the models we used the cross-entropy loss as loss function, and they have been trained for 10 epochs.

It resulted in a grid search that helped us in evaluating and discovering which is the combination that permits the model to have the best training accuracy.
We used a table for reporting all the results, which can be summarized as follows:

1. all learning rate values greater or equal than 3 do not produce good results;
2. all the models with Adam optimizer and learning rate greater or equal than 0.1 reach a local minima and are not able to leave it;
3. Adam optimizer seems to work better with low learning rate values, while the SGD seems to handle higher values better.

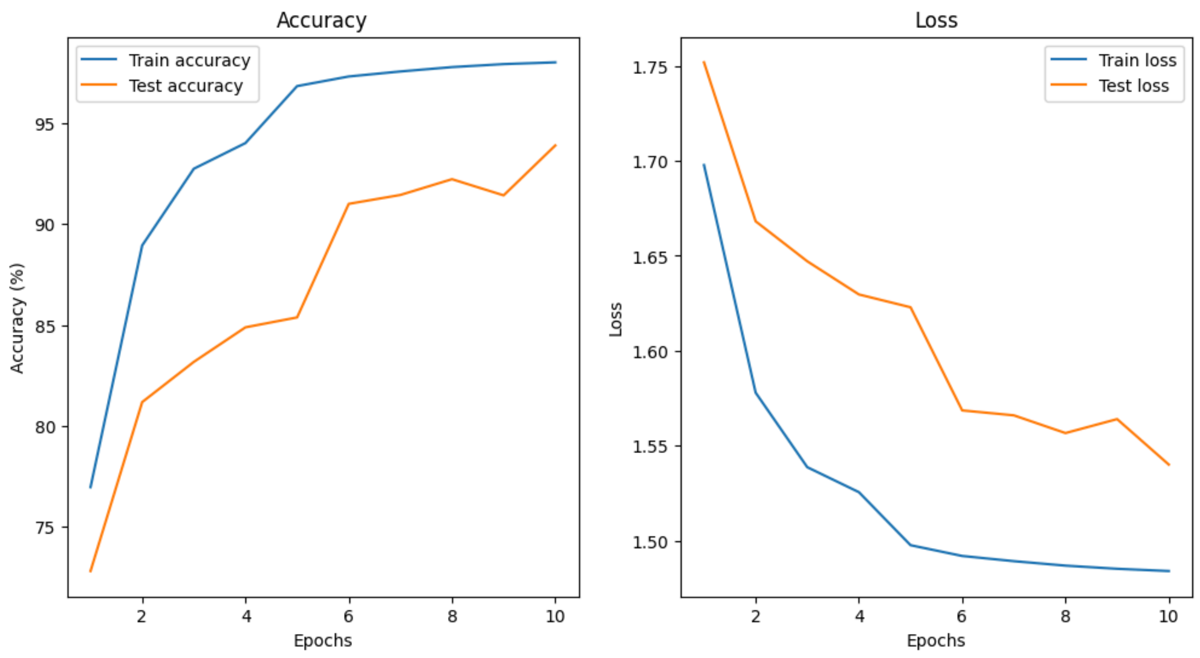
Relevant models

I report below the models we identified as relevant in terms of training accuracy:

- CNN, 3 hidden layers, Adam optimizer and learning rate = 0.001 (98.7% accuracy);
- CNN, 1 hidden layer, SGD optimizer and learning rate = 0.5 (98.9% accuracy);
- FCNN, 1 hidden layer, SGD optimizer and learning rate = 1 (98.8% accuracy);
- FCNN, 1 hidden layer, Adam optimizer and learning rate = 0.0001 (98.7% accuracy);
- CNN, 1 hidden layer, Adam optimizer and learning rate = 0.001 (99.2% accuracy);

4. Testing

We opted to test the CNN with 1 hidden layer, SGD optimizer and learning rate = 0.5 and to plot its accuracy and loss we obtained, firstly in the training set and then in the test set.



It is evident that the accuracy and the loss are inversely proportional, but this was predictable because the more the model is trained, the more accuracy it will have (assuming that the hyperparameters have been chosen correctly), consequently the error it will commit, quantified with the loss function, will also decrease.

Training outcomes

Train Accuracy: 98.0%
Loss: 1.4840

Testing outcomes

Test Accuracy: 93.9%
Test Loss: 1.5401