REAL-TIME HANDWRITTEN CHARACTER RECOGNITION

Artificial Intelligence Master's Degree

Computer Vision & Deep Learning

2023/2024

Menti Tommaso

VR504908

Index

Objective and motivation

In the last few years one field where Artificial Intelligence is used is to read and comprehend any kind of handwritten document. This can be notes taken in class or an ancient book that need to be understood. For this kind of task Computer Vision is fundamental.
For example, in march 2023 the Vesuvius Challenge was launched. It aims to decode carbonized papyrus in Herculaneum during the eruption of Vesuvius in 79 AD and in few months, thanks to the advanced technologies dedicated to this task , many documents were deciphered and translated.
The whole world is going to be digitalized, but to achieve this goal, everything that was written in papers need to be converted in a digital form. In order to do this handwritten recognition becomes essential. Let's think about education, healthcare, historical documents. In all this fields we have millions of handwritten documents that need to be transferred into a computer.
The goal of my project is, in a simpler way, understand a character written in real time.

Pipeline of the project

We can divide my algorithm in few parts:

- Create the model
- Use a dataset found online to fine tune the model
- Create my own dataset
- Train the model with the new dataset
- Write a character and classify it in real time

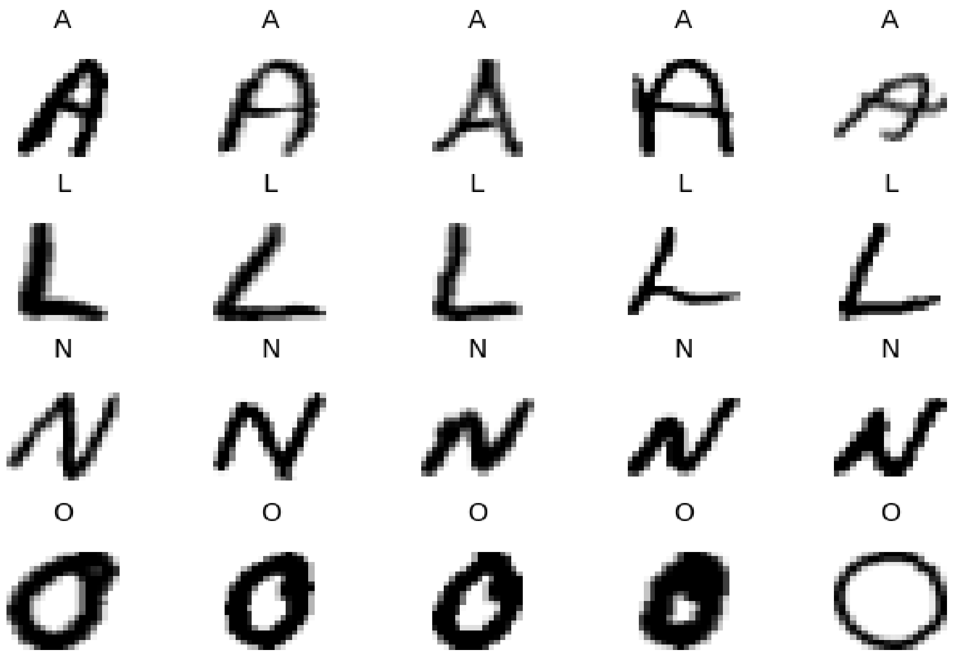
Model

I tried to create a specific CNN for this task.
The model consists of:

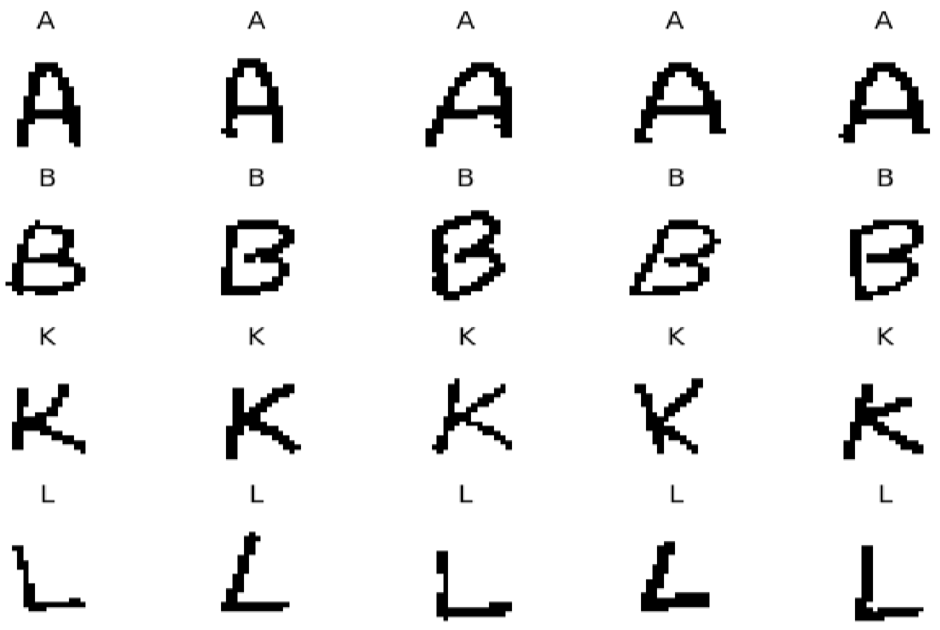
- Convolutional layer (used to extract the features from the image)
    - 1 channel in input (gray image)
    - 32 convolutional filters (3 x 3) with 1 pixel padding
- Convolutional layer
    - 32 channel in input (output of the first layer)
    - 64 convolutional filters (3 x 3) with 1 pixel padding
- Max Pooling layer (used to reduce the dimensionality of the image)
    - kernel: 2 x 2 with stride 2
- Dropout layer (used to prevent the overfitting)
    - probability of 50%
- 2 Batch normalization layers (used to increase the stability and the speed of the training)
- 2 Fully Connected layers
    - ReLu activation function

Dataset

The first Dataset was download from Kaggle.
It consints in a .csv file with 372450 gray scale 28 x 28 images written as an array of pixels.
Each array was saved as an image with the 'SaveImages' function in the 'Dataset_images' folder.



Then, I personally made the second Dataset. I tried to write every single letter multiple times in different ways. I firstly wrote this dataset on a paper, then I took a picture of it and I screenshotted every single letter. I applied a resize and a binarization to each image to eliminate the background's color of the paper (functions: 'process_images' and 'binarization') and save all of them in 'nuova_cartella3'. It contains 248 elements.



Training

Loss Function: ours is classification problem with multiple class so i decided to use a CrossEntropyLoss function. This function measures the difference betwenn the probability distribution of the prediction of our model and the real probability distribution of the class.

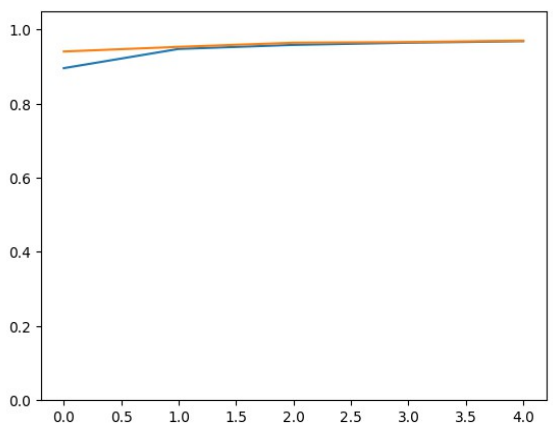
$$\text{Loss} = -\sum y \log(p)$$

Where y is the groud truth label and p is the prediction probability for its class

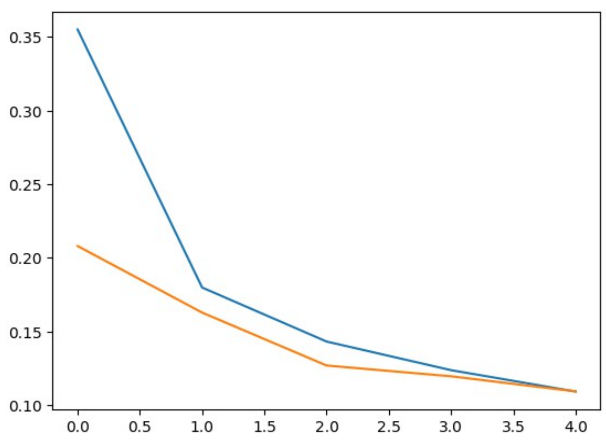
Optimizer: Adaptive Moment Estimation (Adam)

To divide the dataset into train and test I used the train_test_split function of sklearn.
First of all I tried different epochs of training because with such a big dataset I was scared of overfitting. In the end i decided to train the model for 5 epochs for this first training.

Accuracy and Val Accuracy

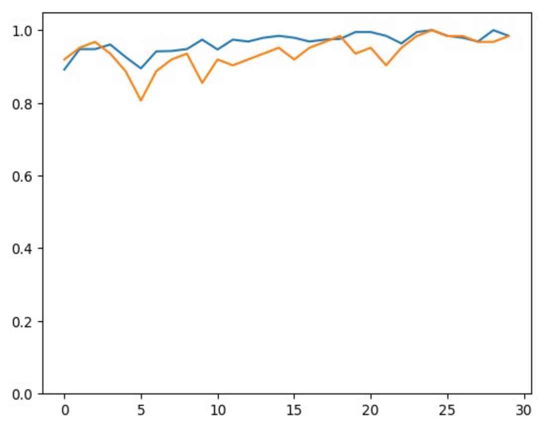


Loss and Val Loss

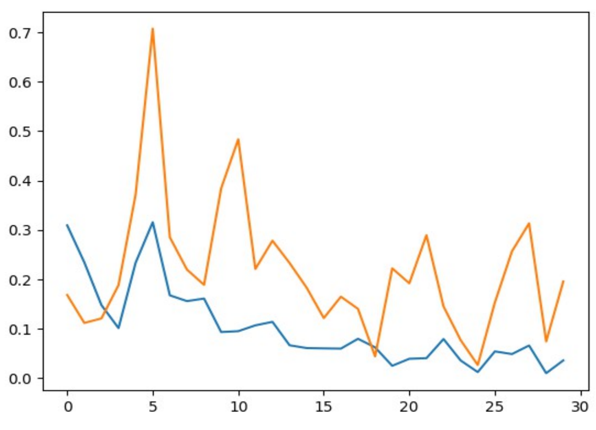


I saved the weights of the model and used them to re-train it with the second dataset.
The Loss function, the Optimizer and the function to split the data are the same.
The second dataset is way smaller than the first one, so i decided to keep 30 epochs.

Accuracy and Val Accuracy



Loss and Val Loss



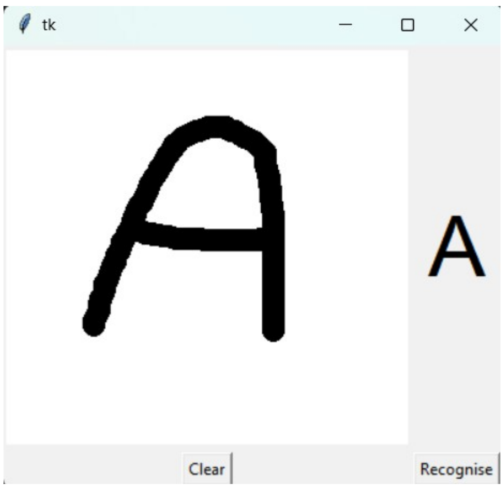
As we can see the Validation Loss in the second training is unstable.
There are possible causes:

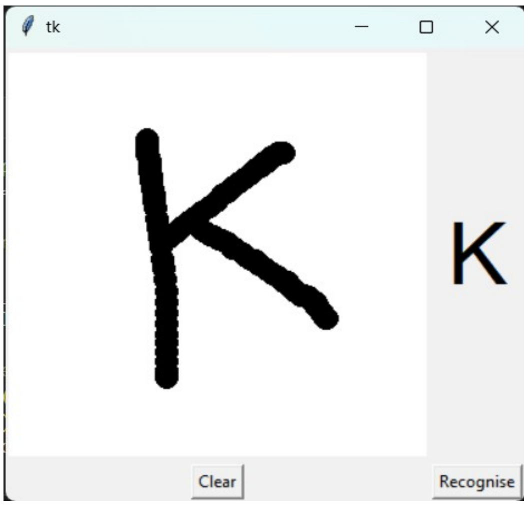
- The main reason (for me) is the small size of the dataset compared to the first one.
- Another possible reason could be the overfitting problem with the first dataset. Two main points make me think this is not the case:
    - Regularization: the Dropout layer in the model
    - High accuracy: as we will see in the evaluation paragraph, the accuracy is still high.

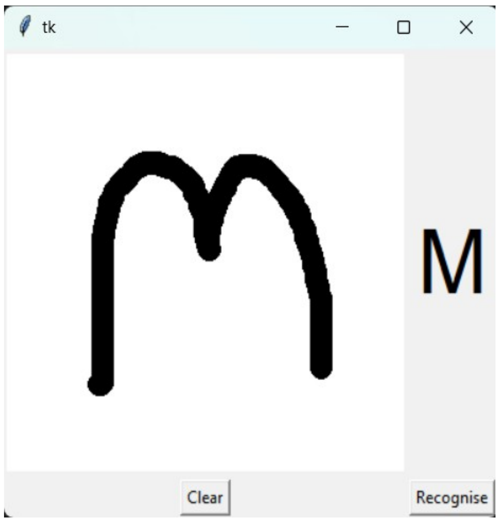
I also tried many different epochs for the first training in order check if it is an overfitting problem and this approach is the best one I have faced.

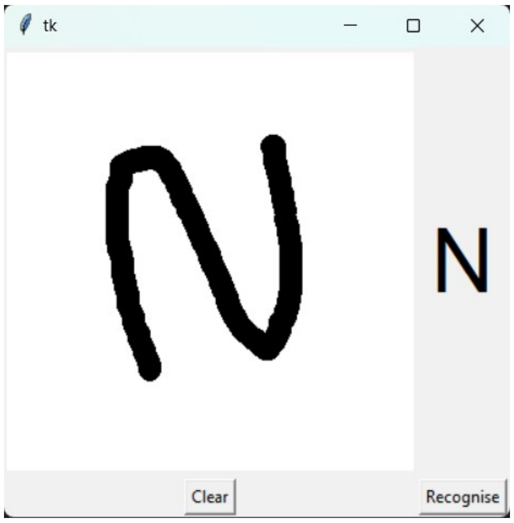
REAL-TIME CLASSIFICATION

I didn't want to create a simple classificator of my handwritten letters with its accuracy, I wanted to make it harder. I decided to create a graphic interface (with Tkinter) that is able to classify in real time handwritten letters. The class 'App' uses the pre-trained model (with the 2 datasets) to understand the character written in the interface and makes its predictions. The prediction with the highest value is the one printed in the window.
Some examples:



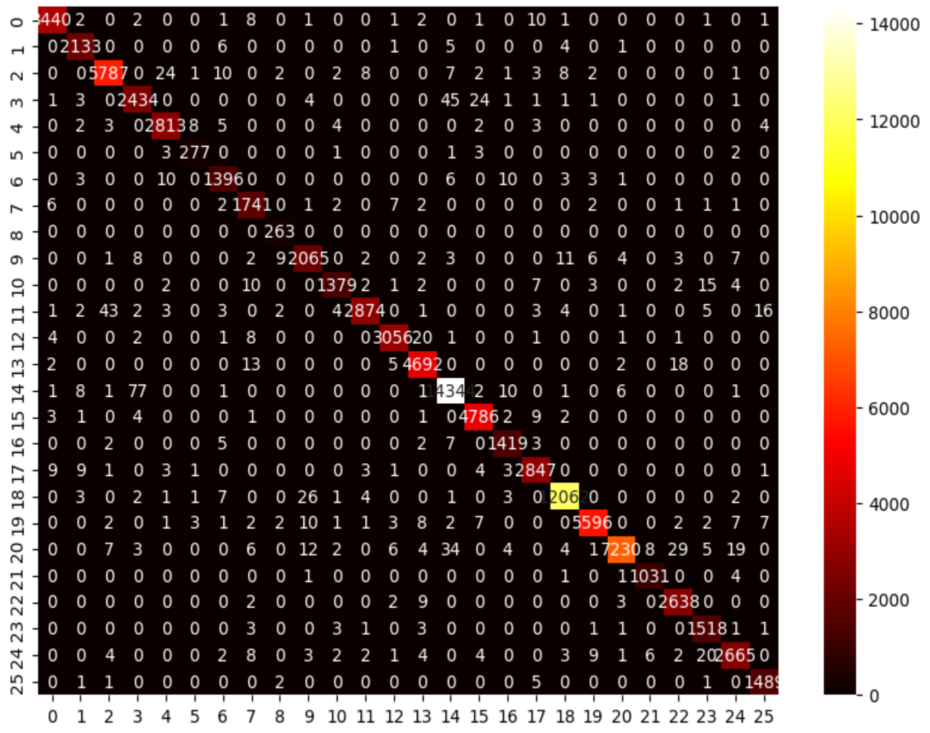




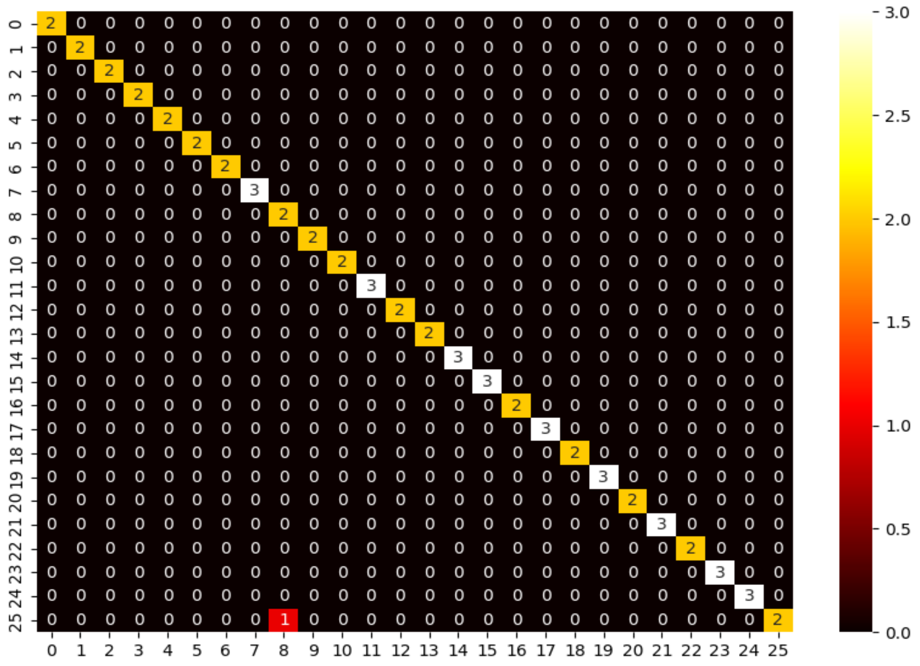


EVALUATION

Accuracy for the first dataset: 98.7778%
Precision: 98.0344 %
Recall: 98.5165 %
Confusion matrix for the first dataset:



Accuracy with the model fine tuned with my own dataset: 98.3871 %
Precision: 98.7179 %
Recall: 98.7179 %
Confusion matrix for my dataset:



Performance analysis

As we have seen in the evaluation part the model performs really well, the accuracy, recall and precision for both trainings are really good. Of course it doesn't perform perfectly because it has to deal with handwritten character, so each of this can be written in infinite ways.
The prediction of the interface is also very accurate, most of the times it outputs the correct character, but sometimes it can be wrong.
When I tested it, for some letters (for example letter A or letter L) it has never failed to predict the correct class, because they have some unique patterns. For some other letters (for example D and O, or W and V) sometimes the model is wrong, that's because it is trained with a large dataset and every letter was written in thousands of ways. These letters have common patterns and sometimes the prediction is not correct.
There is one speciale case, the letter I, that was never been able to predict correctly, the model always predicts the letter T instead.

Conclusion

I have really enjoyed working in this project, my first goal was to develop a handwritten sentence classificator, but I immediatly understood that this task is harder than I thought.
The main struggle of the work is creating the dataset, of course not because it was hard but because it was a really slow and boring process.
In the end the classificator works really well, it's not always correct, but for most of the cases yes.
It only works with capitalized letters but is easily extendable with lower case letters.

References

1. A_Z handwritten Data dataset [https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format]

2. Prediction interface [https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/]