French name transcription from handwriting using deep learning

Carlos Domínguez Becerril

Abstract

Handwriting optical character recognition (OCR) is the process of automatically extracting handwritten information from paper, scans and other low-quality digital documents. The intend of this project is to build a deep neural network that is able transcribe 400.000 handwritten names collected through charity projects to support disadvantaged children around the world.

1 Description of the problem

The task we have to solve consists in transcribing 400.000 handwritten names collected through charity projects to support disadvantaged children around the world. The dataset can be found in Appen machine learning repository [1].

The dataset has the following characteristics:

- The data is divided into a training set (330.961 examples), testing set (41.370 examples) and validation set (41.370 examples).
- The dataset includes three ".csv" files (one for each set) with two columns "filename" and "identity". (see table 1).
  - Filename: The filename of the picture.
  - Identity: The name written in the picture.
- The dataset includes three folders with the pictures associated to each dataset. Each picture has an approximate size of 30x300 pixels.
- The longest name has 34 characters.
- The characters used for each name are the alphabet, blank character ( ), hyphen (-), tilde (‘) and quotation marks (’).

FILENAME	IDENTITY
TEST_0001.jpg	KEVIN
TEST_0002.jpg	CLOTAIRE
TEST_0003.jpg	LENA
TEST_0004.jpg	JULES
TEST_0005.jpg	CHERPIN

Table 1: Example of the testing csv file

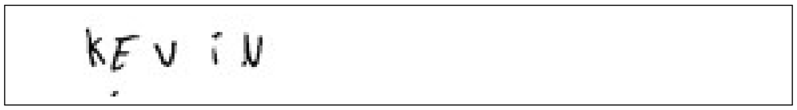


Figure 1: Example taken from the testing data set (TEST_0001.jpg)

As we can see in the figure above the image corresponds to the name "KEVIN".

2 Description of our approach

We organized the implementation of the project according to the following tasks:

1. Data handling
2. Preprocessing of the dataset.
3. Neural network architecture
4. Training and validation

2.1 Data handling

Note: The dataset distinguish between training, validation and testing set. The validation set will be considered as development set in order to ease the naming.

One of the main problems of the dataset is that the number of different names is approximately 100.000, therefore, developing a classifier with this number of labels is not possible due to memory issues.

The approach developed consists in converting each name to a one hot encoding vector. In order to do this, we first create a list of size 34 (the longest name in the dataset) and in each position we calculate the one hot encoding vector for each character which consists in a list of size 31 where all the numbers are 0 except the position of the character that is 1. Notice that the alphabet consists of 30 characters plus an extra character # (hash) for padding, therefore, if our name consists only of 4 characters, 30 more characters are added as padding. The padding is added in order to make all the names the same length. This approach is used to make the neural network learn the features of each character separately instead of learning names.

Example of one hot encoding supposing that the maximum length of a name is 4 and our alphabet is the following one:

"a": 0, "b": 1, "c":2, "#":3

The name "ab" is encoded as:

[1 0 0 0] $\rightarrow a$

[0 1 0 0] $\rightarrow b$

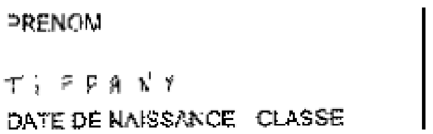
[0 0 0 1] $\rightarrow \#$

[0 0 0 1] $\rightarrow \#$

Our second main problem is that pictures have similar but different sizes (around 30x300) and are in RGB when they only use black and white colors.

The approach developed consists in converting each picture to black and white with a size of 64x256. This size has been decided as it is the closest one that uses powers of 2 (for efficiency) and because most of space used by the name is not even half of width of the picture.

Moreover, some of the pictures have extra text that are not part of the name like "NOM", "PRENOM", "DATE DE NAISSANCE CLASSE"... as seen in figure 2.


PRENOM

T i F F A N Y

DATE DE NAISSANCE CLASSE

Figure 2: Example of picture with extra text

Unfortunately, there is not a pattern we can follow to delete this text and therefore further preprocessing of the pictures is not possible.

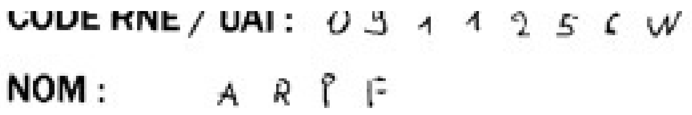
2.2 Preprocessing of the dataset

The preprocessing is an important work when working with datasets. In our dataset we detect two main problems:

- Missing names
- Names format

2.2.1 Missing names

The problem associated with missing names is the existence of pictures where the name of the output is not clear and therefore is labeled as NaN. Moreover, there are names written with just hyphens probably due to some human transcription error.


CODE RNE / UAI : 0 3 1 1 2 5 C W

NOM : A R I F

Figure 3: Picture labeled as NaN (TRAIN_04873.jpg)

We find the following number of missing and incorrect names:

- Training: 565 missing names and 17 incorrect names out of 330961.
- Development: 78 missing names and 0 incorrect names out of 41370.
- Validation: 70 missing names and 2 incorrect names out of 41370.

The decision taken to avoid problems is to delete the examples where the name is labeled as NaN or have just hyphens as it only constitutes a small part of the dataset.

2.2.2 Name format

The problem associated with the names format is that some of them appear in lowercase while others appears in uppercase.

We find the following number of names using upper and lowercase:

- Training: 330366 in uppercase and 13 in lowercase out of 330961.
- Development: 41290 in uppercase and 2 in lowercase out of 41370.
- Validation: 41296 in uppercase and 2 in lowercase out of 41370.

The decision taken to avoid problems is to convert all the names to uppercase.

2.3 Neural network architecture

The neural network used for training our model is a fine-tuned Resnet18 model [5].

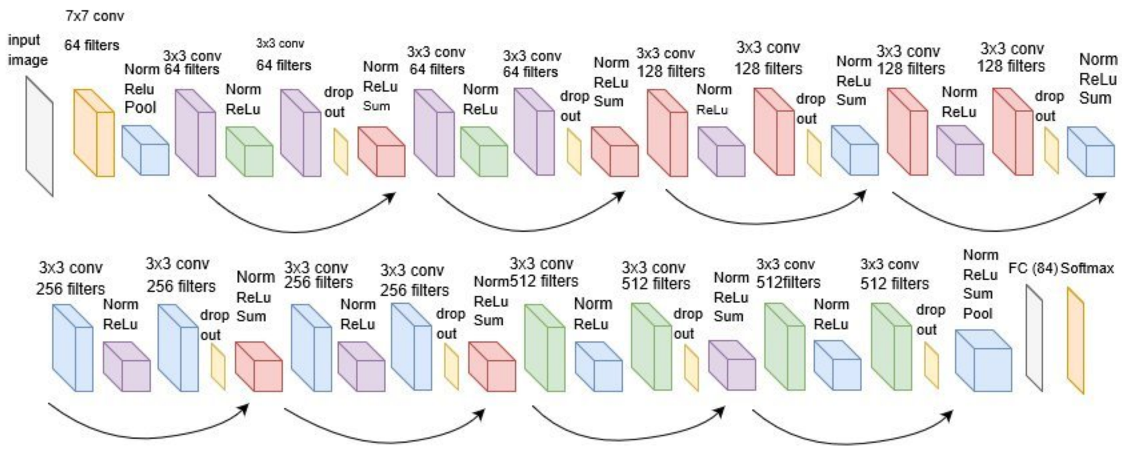


Figure 4: Resnet18 architecture

The following layers are fine-tuned:

- First layer: A convolutional layer is used where the default parameters of resnet18 are used and only the input size of the picture is changed from 224x224 to 64x256.
- Last layer: A dense layer is used where the default parameters of resnet18 are used and only the output features number is changed to 1054 [1].

The last layer will give us the one hot encoding vector as explained in 2.1 Data handling.

2.4 Training and validation

The neural network is trained using the training set and validated using the development set. The test dataset will be used as a hidden validation dataset for reporting the final results.

The hyperparameters for training the neural network are the following:

- Epochs: 10 epochs.
- Batch size: Batch size of 32.
- Optimizer: Adam.
- Learning rate: 1e-4.
- Loss function: The loss is calculated using the multi-label soft margin loss that creates a criterion that optimizes a multi-label one-versus-all loss based on max-entropy, between input x and target y of size (N, C).

In order to validate the neural network, the accuracy of the model is calculated taking into account the two following aspects:

- Correctness of predicted names: A name is considered predicted correct if it matches the ground truth.
- Number of correctly classified characters: Number of characters that are correctly predicted compared to the ground truth.

[1] 1054 comes from doing 31x34 where 31 is the alphabet length and 34 the longest name in the dataset

3 Implementation

The implementation consists in the following steps: preprocessing of the data, data handling, neural network training and validation. All these steps were implemented in Python. Pytorch is used to train the neural network, pandas for handling the data and matplotlib to plot graphs. The implementation of how it works can be found in the Python Jupyter Notebook ”P8_Notebook_Dominguez.ipynb”. The checkpoints of the trained model can be downloaded from [4].

4 Results

The results obtained at the time of training are the following ones:

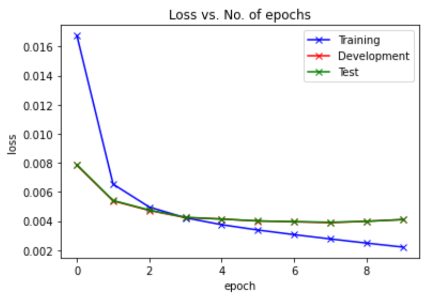


Figure 5: Loss of training, development and testing for each epoch

Epoch	Train	Development	Testing
1	0.016	0.007	0.007
2	0.006	0.005	0.005
3	0.004	0.004	0.004
4	0.004	0.004	0.004
5	0.003	0.004	0.004
6	0.003	0.004	0.004
7	0.003	0.003	0.003
8	0.002	0.003	0.003
9	0.002	0.003	0.003
10	0.002	0.004	0.004

Table 2: Loss of training, development and testing for each epoch

The figure 5 clearly shows the training progress for each epoch. We see that after arriving to the third epoch (second epoch in the figure) the neural network is not able to improve much more staying in a loss of 0.004. Worth noticing how the development and test set overlaps probably due to a high similarity between them. Unfortunately, the loss doesn't give us an accurate information about how the neural network behaves and therefore the following two figures gives us more information as they measure the previous two metrics proposed.

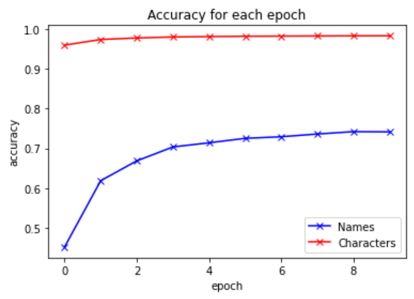


Figure 6: Correctly predicted names and characters (Testing)

Epoch	names	characters
1	0.451	0.959
2	0.618	0.973
3	0.668	0.977
4	0.703	0.980
5	0.713	0.981
6	0.725	0.981
7	0.729	0.982
8	0.736	0.982
9	0.741	0.982
10	0.741	0.983

Table 3: Correctly predicted names and characters (Testing)

The figure 6 clearly shows that after the third epoch there is a considerable improvement when we calculate the correctly predicted names obtaining an accuracy of around 74%. The accuracy of characters can be considered misleading as it is calculated considering that each name has 34 characters and most part of it is padding. Nevertheless, this metric ensures that it doesn't predict more characters than the ones that are in the picture as we obtain a value near 98%.

5 Conclusions

In our project we have made use of a fine-tuned Resnet18 neural network to solve the handwriting optical character recognition introduced in [2]. The method proposed evaluates the neural network using two metrics: correctly predicted names and correctly guessed characters.

The results show that using this neural network it obtains an accuracy of around 74% predicting correctly the names that appear in the pictures while obtaining a 98% when guessing the correct characters. These results can be considered good enough but not comparable to humans due to not being able to guess correctly almost a 30% of the dataset. Improvements can be done as proposed by [3] that makes use of a combination of convolutional neural networks and bidirectional LSTMs and where the loss is calculated using the Connectionist Temporal Classification loss (CTC).

References

[1] Appen Limited Handwriting Recognition dataset 2017. Dataset

[2] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. Neural computation, 22(12):3207–3220, 2010. Paper

[3] Marcin Namysl, Iuliu Konya. Efficient, Lexicon-Free OCR using Deep Learning. 2019. AarXiv:1906.01969. Paper

[4] Carlos Domínguez. French name transcription from handwriting using deep learning. 2020. Checkpoints

[5] Mujadded Al Rabbani Alif, Sabbir Ahmed, Muhammad Abul Hasan. Isolated Bangla handwritten character recognition with convolutional neural network. 2017. ICCIT. Paper