Name identification using ANNs

Iker García and Eritz Yerga

Abstract

In this document we report our proposal for the application of name identification using ANNs. The classifiers that we have selected for the classification tasks were: A pipeline of a RBM to extract the features and MLP to classify, a MLP using HOG (Histogram of Oriented Gradients) features, a MLP using PCA (Principal Component Analysis) and a MLP. We have implemented the classification process using the scikit-learn library. We have learned the classifiers using the train data and computing various metrics in the test data. From our testing the best results were produced by the MLP with PCA features and the MLP.

1 Description of the problem

The goal of the project is to solve the task of name transcription from handwriting images implementing a NN approach and using a database with a large number of images of handwritten names [1].

The dataset includes over 125,000 images of handwritten names along with human contributors' transcription of these written names. Most names on the dataset are in French, which means that they can include accent marks.

The database has the following characteristics:

- 1 unique identifier per entry on the database.
- 1 url to the corresponding image per entry.
- 1 transcription of the name per entry.
- 1 label indicating if its a first name or a last name per entry
- The images don't contain only the names and sometimes contain more data that needs to be cut or deleted, the transcriptions of the names sometimes are missing, are not correct or are repeated in multiple lines.

Since we have the transcription of the names this is a supervised learning problem.

2 Description of our approach

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

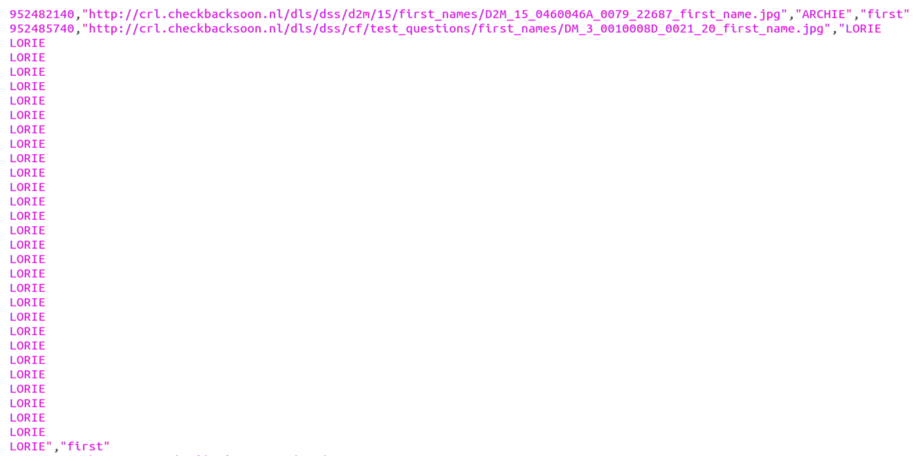
1. Preprocessing of the dataset
2. Preprocessing of the images
3. Feature extraction
4. Classifiers
5. Inference
6. Validation

2.1 Preprocessing of the dataset

The original dataset has many format problems, so the first step was to solve this. We have two types of images, some in which the name is written next to the NOM or PRENOM word and others which the name is written under that words.

Two examples of the different type of images in the dataset

In the second type of images the transcribed name is duplicated in many lines, there are also labels that are duplicated only in the line below that belong to the first type of images.



```
952482140,"http://crl.checkbacksoon.nl/dls/dss/d2m/15/first_names/D2M_15_0460046A_0079_22687_first_name.jpg","ARCHIE","first"
952485740,"http://crl.checkbacksoon.nl/dls/dss/cf/test_questions/first_names/DM_3_0010008D_0021_20_first_name.jpg","LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE
LORIE","first"
```

One of the duplicated entries in the dataset

We solved this with a small program made in Java (database_fix.java inluded in the GitHub repository [2]) that deletes the duplicated names and modifies the last label to indicate if the name is positioned below the NOM/PRENOM word or next to it. In case the name is located on the right the last label will have the value ”first” or ”last” and in case that the name is located below it will have the value ”first_b” or ”last_b”. The dataset obtained from this modifications can be found on our GitHub repository [2] with the name first_and_last_names_fix.csv.

2.2 Preprocessing of the images

We used the pandas library to import the dataset from the CSV file. The dataset does not contain the images itself so we need to download them.

In the images apart from the name that we want to transcribe we have a lot of noise that we want to erase, for example the code of the image, the word NOM: or PRENOM: or some splitted characters that are part of other names. NOM or PRENOM words appear in all the images and because they are written in a machine and then printed they have always the same exact form, so we extracted as reference the last two characters of this words from one image of the database. We used a template matching algorithm from the OpenCV library [3] (chapter 7: Histograms and Matching) [4] using this reference image that locates the NOM or PRENOM word. After this is done, we use the information about the location of the word to crop the image and extract the name. However, with this procedure apart from the name we can also crop some noise like parts of other names written above or below. Because the images are taken with the name centered, it would never touch the corners so to remove the noise we find any figure that is touching the lower border of the image and we delete it.

After this what we have is a image that contains only the name that we want to recognize. The next step of the procedure if the character extraction. This is done with a clustering algorithm. First we binarize the image, then using clustering we calculate the different connected components of the binarized image [5] [6]. When this is done we calculate the coordinates of every components to extract the characters, but the problem is that not every character must be a unique component, for example the point of an ”i”, of one of the lines of an capital ”E”, so for calculating this coordinates we take in account that if a component is above other component both of them belong to the same character. The characters then are extracted and rescaled to a 28x28 image, this are the images that we will use to train our Neural Networks.

This procedure does not always work correctly, sometimes we obtain a different number of characters that the number of letters that the name has in the label, this can happen due to noise, characters written splitted, a bad label of the name... Nevertheless we are able to obtain the same number of characters that the name label has with approximately an 94% of success rate. The remaining 6% of the names are not added to the train or test data, because we know that we will train with data that is wrong and we also know that if we try to test the result with it, it would be a labeled as failure. This inconsistent names are added to another list so after having trained the neural networks we can try to extract some valuable information from them, for example try to test with of the images extracted are characters and which are noise so we can add to the database the images that we recognize as characters.

We have no way of checking if the character extraction has been done properly in the data that we will use for training and testing, but based on the results of the project we think that we have a enough amount of properly extracted characters to successfully train our Neural Networks.

The data is splitted into two different sets: train and test, for validation. We use the same train set to learn all the classifiers, and the same test set for evaluating their accuracy.

2.3 Feature extraction

We have implemented different types of feature extraction to test which of them gives the best results [7].

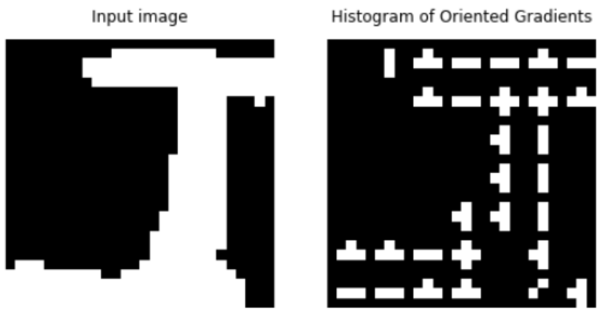
2.3.1 Restricted Boltzmann Machine

The Restricted Boltzmann Machines contains a set of latent variables that represent higher order patterns present in the data. They are are commonly used in the character recognition task. This is an example of the features learned by our RBM:

Features learned by the RBM

2.3.2 Histogram of Oriented Gradients

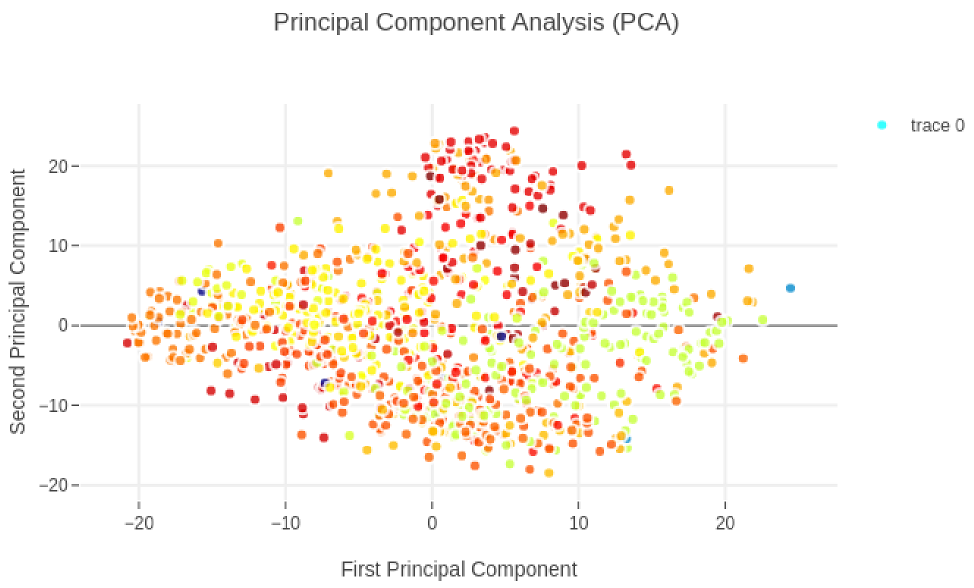
This technique uses the distribution of directions of gradients as features [8]. Gradients of a image are useful because the magnitude of gradient is large around edges and corners which gives us a lot of information about objects shape. The problem with this method is that our characters are a 28x28 image, so they could not be big enough to take advantage of all the potential of this technique. This is an example of feature extraction using ”HOG” of one of the characters in the dataset.



Original character and the result of HOG

2.3.3 Principal Component Analysis

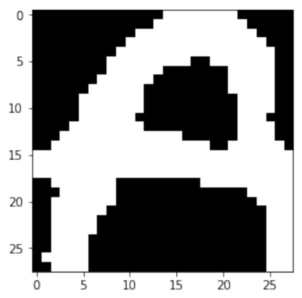
PCA is a linear transformation algorithm that seeks to project the original features of our data onto a smaller set of features while still retraining most of the information. This algorithm tries to find the most appropriate direction/angles that maximize the variance in the new subspace [9] [10]. The number of features used in our project was 100.



PCA of the first two features

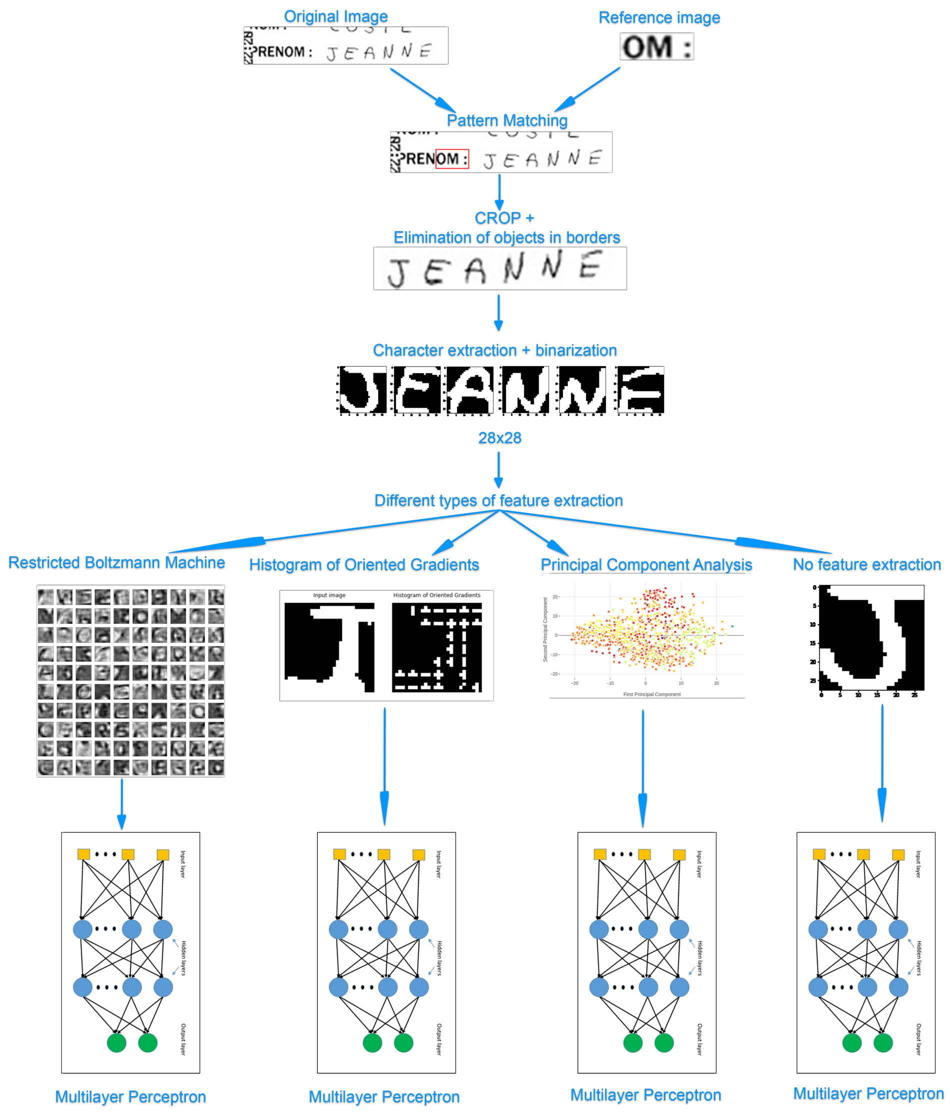
2.3.4 No feature extraction

For the purpose of evaluating the previous methods we also trained our neural networks without any type of feature extraction, that is, using the binarized image of the characters.



A binarized character without feature extraction

This image represent the process that we follow to train our network


Original Image
Reference image
Pattern Matching
CROP +
Elimination of objects in borders
Character extraction + binarization
28x28
Different types of feature extraction
Restricted Boltzmann Machine
Histogram of Oriented Gradients
Principal Component Analysis
No feature extraction
Multilayer Perceptron
Multilayer Perceptron
Multilayer Perceptron
Multilayer Perceptron

Process diagram

2.4 Classifiers

We use four different classifiers with the following parameters:

1. Pipeline of RMB for feature extraction and MLP for classification:
   - RMB:
     - n_components = 300
     - learning_rate = 0.01
     - n_iterations = 45
   - MLP:
     - layers = (300, 400, 150)
     - activation_function = 'relu'
     - max_iterations = 5000
     - tol = 0.0001

2. MLP classifier with HOG (Histogram of Oriented Gradients) features:
    - layers = (300, 400, 150)
    - activation_function = 'relu'
    - max_iterations = 5000
    - tol = 0.0001
3. MLP classifier with PCA (Principal Component Analisis) features:
    - PCA:
        - n_components = 100
    - MLP:
        - layers = (300, 400, 150)
        - activation_function = 'relu'
        - max_iterations = 5000
        - tol = 0.0001
4. MLP classifier with character images directly:
    - layers = (300, 400, 150)
    - activation_function = 'relu'
    - max_iterations = 5000
    - tol = 0.0001

The learning algorithm for the RBM is the Stochastic Maximum Likelihood and learning algorithm for all the MLPs is the default in Scikit-Learn: Adam optimizer.

2.4.1 Rationale behind the conception of the NN

After some research we found that multilayer perceptron was the most popular network for this task and that it commonly produces good results, which is why we chose this type of Neural Networks. There are other more advanced methods used for handwritten text recognition such as convolutional neural networks, but the objetive of this project was solve the problem using an NN aproach excluding Deep Learning.

2.5 Inference

Once the Neural Networks are trained the process to predict new names is very similar to the process to train it. The new image must follow the same process that the images used for training, that means, we need to extract each individual character on the image. Then, apply the feature extraction that we want to use, for example calculate the Histogram of Oriented Gradients or calculate the state of the RBM for this image. Then the multilayer perceptron predicts a letter for each individual character. The final step is to put the values predicted by the Neural Network together to form the name.

2.6 Validation

To validate our results we compute the scikit classifier metrics and our own full name prediction metrics in the test data. Another possibility was to compute the cross-validation in the complete dataset but we used the split between train and test because it was simpler. We divided the dataset in 80% train 20% test batches.

We computed the classification report of Scikit metrics (which gives the precision, recall and f1-score for each classifier) for individual character prediction. We also tested full name recognition and we output the full correct name ratio and the correlation ratio (how similar all predicted names are to the original label) for each classifier.

3 Implementation

All the project steps were implemented in Python. We used pandas and urllib for reading the original database and getting the images from the servers, scikit-image, cv2 (python3-opencv) and scipy for preprocessing the dataset, matplotlib and plotly for plotting different data and scikit-learn for the classification tasks. We illustrate how the implementation works in the Python notebook `Handwitten-Name-Recognition-ANNs.ipynb`.

3.1 Usage

Downloading all the database and training the Neural Networks takes a long time, so in the GitHub repository of the project [2] we added a database with 10.000 images downloaded and pre-trained classifiers with that database. At the beginning of the notebook there are some global variables that allow the user to choose how to run the program.

1. dataset_load_method: If the value is 'load' it will load the database provided (the included .7z file needs to be extracted, check the README.md file in the Notebook folder for more info). If the value is 'download' it will download the images (the number of images that are going to be downloaded can be modified in the call to gen_dataset() function).

2. save_database: If the value is true it will save to a file the generated dataset with the downloaded images.

3. load_classifiers: If the value is true it will load the pre-trained classifiers provided (the included .7z files need to be extracted, check the README.md file in the Notebook folder for more info). If the value is false it will train the classifiers.

4. save_classifiers: If the value is true it will save to a file the classifiers once they have been trained.

5. save_results: If the value is true it will save the results of the classification test to a file. It's useful because when dealing with a lot of data the RAM fills up and could cause the screen to freeze.

6. enable_error_output: If the value is true it will print some information useful for debugging and for improving the program such as the images where the character extraction has failed (if the value is false it will still print the % of the images where the character extraction worked correctly).

4 Results

We ran our testing for the first 30000 names in the dataset because testing with more data was not possible with our machines (we don't have enough RAM), the included prebuilt dataset and classifiers from our repository were from this testing.

The precision produced by the RBM-MLP Pipeline, MLP with HOG features, MLP with PCA features and MLP classifiers in individual character recognition were, respectively: 0.92, 0.89, 0.92 and 0.93.

The full name correct ratio produced by the RBM-MLP Pipeline, MLP with HOG features, MLP with PCA features and MLP classifiers were, respectively: 0.664, 0.579, 0.688 and 0.709.

The full name correlation ratios (similarity of name produced to original name) produced by the RBM-MLP Pipeline, MLP with HOG features, MLP with PCA features and MLP classifiers were, respectively: 0.923, 0.899, 0.929 and 0.934.

Therefore, the best classifiers were the MLP with PCA features and the MLP.

The results of the computation of the Scikit metrics for the RBM-MLP Pipeline, MLP with HOG features, MLP with PCA features and MLP classifiers are respectively shown in Tables bellow.

Train	Names:	23,945	Characters:	131,759
Test		5,987		33,211
Precision	RBM + MLP	HOG + MLP	PCA + MLP	MLP
-	0.50	0.70	**0.74**	0.70
A	**0.94**	0.91	0.93	**0.94**
B	0.85	0.76	**0.86**	0.78
C	0.92	0.90	**0.93**	0.92
D	0.86	0.80	**0.88**	0.86
E	0.94	0.93	0.95	**0.96**
F	**0.70**	0.66	0.65	0.64
G	0.86	0.83	**0.91**	0.90
H	0.84	0.81	**0.88**	0.87
I	0.91	0.90	0.91	**0.93**
J	0.88	0.89	**0.90**	0.89
K	0.85	0.75	**0.90**	0.88
L	**0.96**	**0.96**	**0.96**	**0.96**
M	0.87	0.84	0.86	**0.89**
N	0.92	0.90	**0.95**	0.94
O	0.91	0.90	0.91	**0.94**
P	0.81	0.72	**0.85**	0.78
Q	**0.86**	0.62	0.84	0.81
R	0.89	0.83	**0.92**	0.91
S	0.93	0.92	**0.95**	**0.95**
T	0.93	0.91	0.92	**0.94**
U	0.92	0.91	0.93	**0.95**
V	0.88	**0.89**	0.82	0.88
W	**0.90**	0.68	0.81	0.83
X	**0.94**	0.82	0.91	**0.94**
Y	0.93	0.86	**0.94**	0.93
Z	0.82	0.83	**0.93**	0.87
TOTAL	0.92	0.89	0.92	**0.93**

Individual character classification results

We can also see the full name correct ratios and correlation ratios in the table bellow.

Classifier	Correct percentage	Correlation ratio
MLP with RBM features	0.664	0.923
MLP with HOG features	0.579	0.899
MLP with PCA features	0.688	0.929
MLP only	**0.709**	**0.934**

Full name prediction results

As we can see, the percentage of correctly classified names is not too high, but if we calculate the correlation of the predicted names with the real names the number is really high, that means that what the classifier predicts is very similar to the real name, so the names that are not correctly classified are probably wrong in very few characters.

5 Conclusions

In our project we applied a multilayer perceptron combined with different types of feature extraction to the "Transcriptions of names from handwriting" dataset. We have computed the accuracy of this Neural Networks and we observed that surprisingly the simplest implementation, the MLP without feature extraction produces the highest accuracy.

Finally as future upgrades in this implementation we did not do anything with names where the character extraction gives us inconsistent results. Even so the program stores in a list all this names to make possible to work with them in the future. Due to computational limitations we were not able to train with the full database, training with more names could help to improve the results. Also fine tuning of the parameters of the neural networks could improve the results but computational power is again a limitation.

References

[1] CrowdFlower. Data for everyone library: Transcriptions of names from handwriting [`https://www.crowdflower.com/data-for-everyone/`], 2016.

[2] Iker García (ikergarcia1996) and Eritz Yerga (eritzyg). Github repository: Handwritten names recognition [`https://github.com/ikergarcia1996/Handwritten-Names-Recognition`], 2017.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[4] OpenCV. Opencv documentation: Template matching [`https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html`].

[5] Scipy Lectures. Labeling connected components using scikit-image: [`http://www.scipy-lectures.org/packages/scikit-image/auto_examples/plot_labels.html`].

[6] Scikit-Image. Scikit-image documentation: skimage.measure.label [`http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label`].

[7] Dewi Nasien Muhammad 'Arif Mohamad, Haswadi Hassan and Habibollah Haron. A review on feature extraction and feature selection for handwritten character recognition. *International Journal of Advanced Computer Science and Applications(ijacsa)*, 6(2), 2015.

[8] Satya Mallick. Learn opencv: Histogram of oriented gradients [`https://www.learnopencv.com/histogram-of-oriented-gradients/`].

[9] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[10] Anisotropic. Interactive intro to dimensionality reduction [`https://www.kaggle.com/arthurtok/interactive-intro-to-dimensionality-reduction/notebook`], 2017.