Ibn Tofail University

National school of applied sciences

Department of computer science, logistics, and mathematics

An End-to-end Offline Text-independent Writer Identification system deployed as a web API

Author:
Hicham DAHMOU
dahmou.hicham.0@gmail.com

Supervisors:
Aniss MOUMEN
Ahmed REMAIDA
Benyoussef ABDELLAOUI

November 5, 2020

Abstract

Our work is a research on the problem of "writer identification based on handwriting", for which we used diverse unsupervised learning approaches (clustering, Neural Networks, dimensionality reduction) and computer vision algorithms, all combined on a modular testing pipeline for the experts (includes 8 modules organized on 5 steps) and a Web API that offers the resulting model as an easy to query service plus a face recognition service and management of the history of past operations stored on a non-relational database.

While the approaches we used are diverse, we prioritized the ease of maintenance of the pipeline (created from scratch), which paid off in the final stages of the project where adding a new module or datasets is feasible.

The source code and the latest version of the report can be found here:

https://github.com/Nagisa-sys/WriterIdentification

Contents

Chapter 1

Introduction

1.1 Motivation

With the high numbers of discovered hacked accounts each day and the regulations that always require more assurance about an individual identity before allowing access to a resource[1], multi-factor authentication is a natural solution to ensure such guarantees[2].

In contrast with classical authentification methods based on a password or physical metrics (like fingerprints and the shape of the face), behavioral biometrics: "traits and micro-habits that could be used for authentication", provide continuous authentication, that works passively without users stopping their normal activity, which spare them the frustration of MFA and provide a good balance between security and usability [3] [4].

And here comes the role of writer identification and retrieval that belongs to the group of handwriting biometrics, which itself is a part of the larger group of behavioral biometrics:

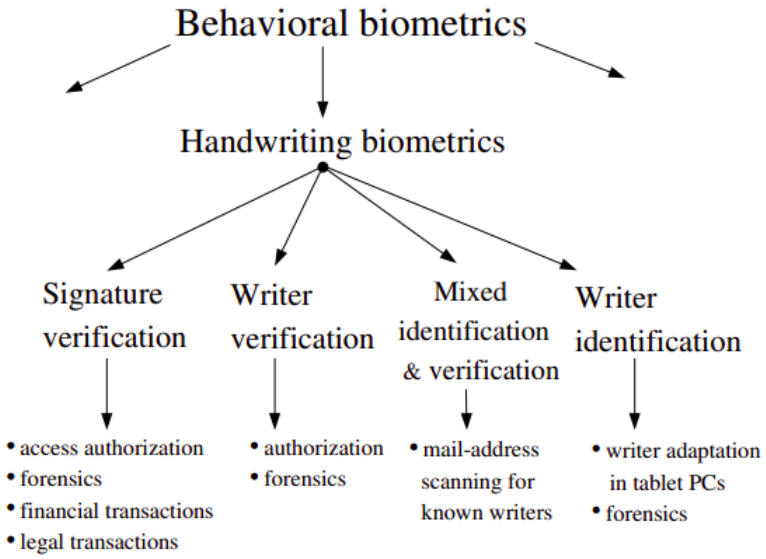


Figure 1.1: Types of handwriting biometrics and application areas [5]

We will focus on the writer identification task, where based on an unknown sample as input, we identify the author among a set of labeled documents with their authors:

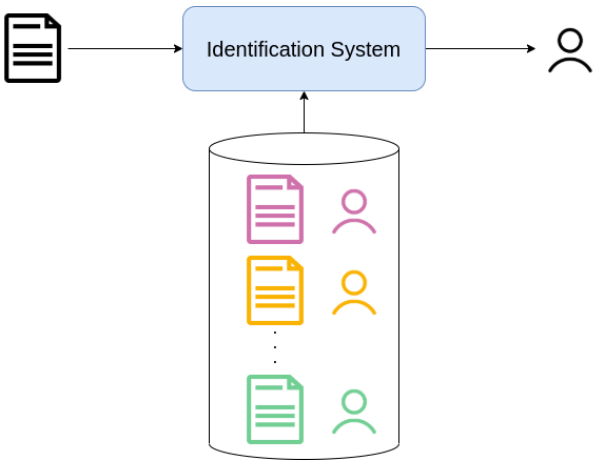


Figure 1.2: Writer identification System

The other mode: Writer verification could be seen as an identification task where we check the coefficient of similarity, this change could be accomplished by minimum effort, therefore, we prefer that the user pick and choose based on his use case.

Speaking of use cases, some of them for writer identification task:

- In the court of justice, when there is a need to authenticate a will or in criminal investigations, [6]
- In the domain of studies of ancient inscriptions/documents: Palaeography, where the classification of shapes used by scribes helps dating historical manuscripts[7].

Note: No offense, but I didnt include graphology: the study of handwriting from all angles, including its relationship with character or personality[8] even if it sounds exciting but it is generally considered a pseudoscience [9].

A famous example where writer identification (done manually) was used is in the case of the Kidnapping of Charles Lindbergh in 1932, where after examining and eliminating more than 2 million samples of handwritten notes, they narrowed down the kidnapper[10] (Imagine if they had an automatic system that could do it).

1.2 Individuality of handwriting

If we model our system as a single f function, it's input will be the image of handwritten text, and its output is the writer id; for such a system to function correctly, the f function must be at least subjective (I am using a loose notation so take it easy).

Such condition is satisfied if and only if the writer is unique for each handwritten sample, if that's not the case, the system won't be feasible and we will end the project here (the condition is satisfied, look at the size of this report).

The answer such a question we ask domain experts: the FBI Laboratory [11] that approved handwriting analysis a scientific act based on the following three foundations :

- Individuality: No two writers share the same combination of handwriting characteristics given sufficient quantity and quality of writing to compare (also apply to identical twins living together),
- Variation: No one person writes exactly the same way, even within several repetitions of writings,
- Writing Skill: Every writer has a writing skill that cannot be dramatically improved in a short time frame while maintaining all appearances of natural writing.

Actually, in the USA there is a standard for the forensic document examination, submitted and maintained at a certain time by the ASTM committee, and now by SWGDOC[12].

The one limitation of this process is that imitation can be detected by experts, but not attributed to the person who traced the text/signature.

1.3 Project context

This project is a part of a larger system that provide the services of writer identification (based on handwriting of Latin characters) and face recognition (using a pre-existing package) to non-expert users:

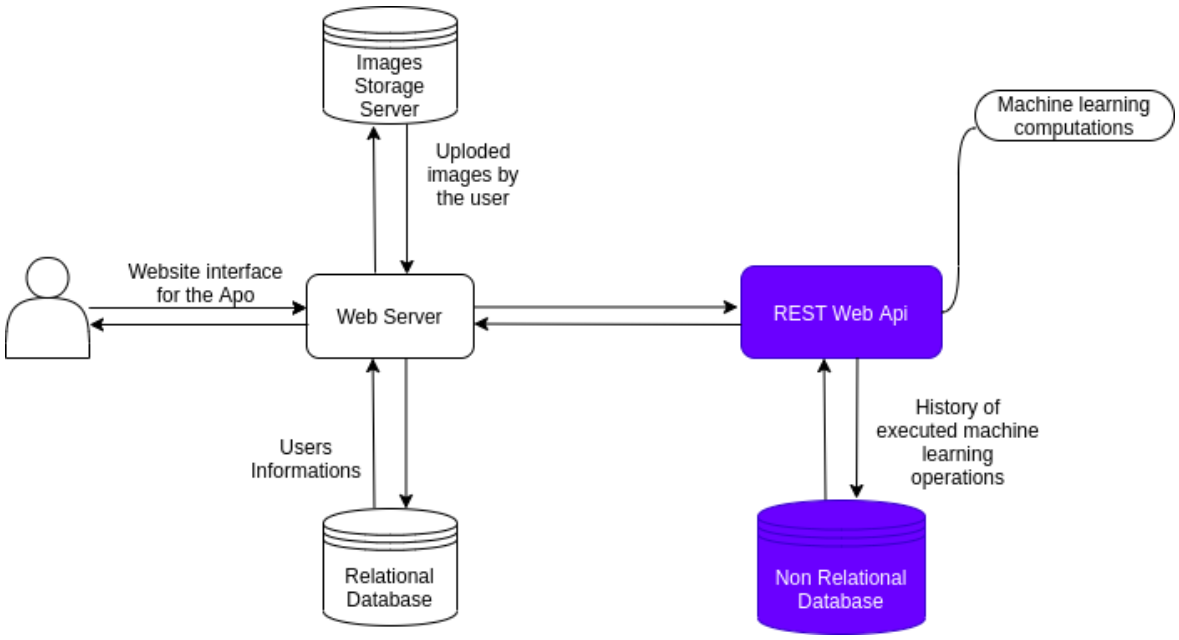


Figure 1.3: The big picture of the project's context

Our part is to manage the conception, implementation, testing, and deployment of writer identification and face recognition services via a REST API and generating a log of past operations, which could be used to provide indicators for the health of the global system and as a source of data for future training sessions.

Taking into account that face recognition is a mature field of research, and the time limitation, we will concentrate our efforts on writer identification and leave the other one to pre-build packages.

1.4 Project objectives

A wise man once said: "Its not bad to dream, But you also have to consider whats realistic!"; another one: "Progress and innovation happen when you set unrealistic goals"; but the one that I agree with the most is: "There are no unrealistic goals, only unrealistic deadlines".

Considering the deadline, and having the motivation, we state that we want to at least:

- Get a general grasp on the state of the art methods used for writer identification and computer vision in general, theoretically and practically,
- To go through the complete process from research about the problem to deploying a functional service,
- To be familiar with the tools used for artificial intelligence and computer vision,
- Write in English a solid report that documents the whole process and publish the entire project on a public platform.

1.5 Document structure overview

In this chapter, we already tackled the feasibility, context, motivations, and objectives, as for the rest of the report:

- Chapter 2 gives an overview of the different approaches taken by researchers to identify the writer of a handwritten document. We also present some competitions in this domain, the databases they used, and a variety of articles for curious readers,
- Chapter 3 present the encoding pipeline that will be applied to documents as part of the approach we have taken to identify the writer, go into details for each module used, the different possible combinations and data structures that will be used to reduce time complexity,
- Chapter 4 present the technologies used for the implementation and the resulting jupyter notebooks,
- Chapter 5 present the benchmarking datasets, evaluation protocol, how we have done the tuning of the hyperparameters, and finally the different experiments that we concluded on the datasets using the pipeline we created,
- Chapter 6 is about the web API that we have developed to offer writer identification and other services to our client,
- Finally, chapter 7 is where we provide recommendations for readers who want to build on our work.

Chapter 2

Writer identification: field state

2.1 Introduction

Automatic writer identification is a field of research that makes use of computer vision and pattern recognition to identify the writer of a document, given a set of known writers, which can be done in many modes and using different approaches.

2.1.1 Off-line vs On-line

Depending on the input method of writing we can distinguish between the two modes.
In on-line writer identification, we rely on timed information about the writing process, like the speed, angle, or pressure, which are not available in the off-line mode [13] [14]where we rely solely on the final result: an image of handwritten text.
It's like writing a note using an intelligent pen that records the angle, pressure on the screen, and on itself, speed: all of it in a time series VS presenting just the image of the note.
Off-line writer identification is tackled using two different approaches:

2.1.2 Allograph-based vs Textural-based

Allograph-based methods rely on local descriptors computed from small letter parts (allographs). Subsequently, a global document descriptor is computed by means of statistics using a pre-trained vocabulary. In contrast, textural-based methods rely on global statistics computed from the handwritten text, e. g., the ink width or angle distribution. Therefore, a texture image is first transformed into a pool of local features, which are then aggregated into a global representation for an entire image or patch. Both methods can be combined to form a stronger global descriptor[15].

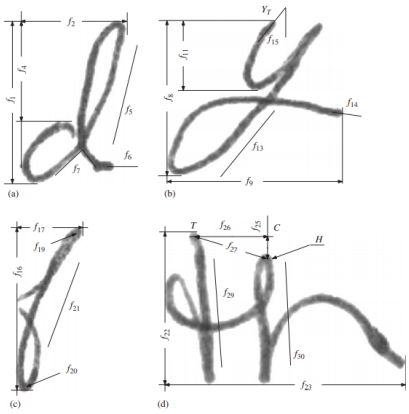


Figure 2.1: Some of the features extracted as part of an allogrph-based method[16].

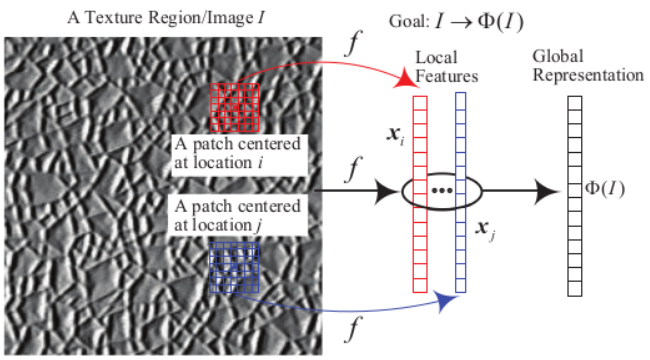


Figure 2.2: The goal of texture representation is to transform the input texture image into a feature vector that describes the properties of the texture, facilitating subsequent tasks such as texture recognition[17].

2.1.3 Text-dependent vs Text-independent

Another common classification is text-dependent VS text-independent automatic writer identification, as the name implies one is based on character comparisons to boost accuracy, therefore, requires that the same text is used by all the writers and the other is more general by being independent of the text.

2.1.4 Multi-script identification

Additionally, because of the differences of character sets of different languages (for example, Arabic, Latin, Japanese), models on multi-script identification naturally, have low identification rates [18] compared to others designed with just one set of characters in mind, therefore, the majority of models are language-specific or treat a small subset of close languages[**?**].

2.2 ICDAR and ICFHR competitions

ICDAR is the International Conference on Document Analysis and Recognition is an international academic conference which is held every two years in a different city. It is about character and symbol recognition, printed/handwritten text recognition, graphics analysis and recognition, document analysis, document understanding, historical documents and digital libraries, document-based forensics, camera, and video-based scene text analysis[19][20].

In the same spirit, there is ICFHR: International Conference on Frontiers of Handwriting Recognition which is also interested in handwriting recognition and its applications[21].

Apart from the conferences, the committees support a set of competitions that address current research challenges related to areas of document analysis and recognition including writer identification, therefore these competitions are a good opportunity to get to know the cutting edge techniques in our research subject, especially that the last challenge was in organized 2019 by ICDAR.

The winner in each challenge based on the accuracy of the single labeling task:

2.2.1 ICFHR2016-CLAMM

ICFHR2016 Competition on the Classification of Medieval Handwritings in Latin Script

- **Dataset Description** : Gray level images, annotated with a single script class from 12 classes representing different styles of character shapes used by scribes in producing manuscripts in Europe in the years 500 C.E to 1600 C.E[22].
- **Training/Testing size** : 2000/1000 pages.
- **The Winner title** : Unsupervised Feature Learning for Writer Identification and Writer Retrieval[23].
- **Accuracy** : 83.90%.
- **Features learning** : Part handcrafted (SIFT) part Unsupervised (NN).
- **Lazy/Eager learner** : Eager.
- **Summary** : First SIFT descriptors are computed on the training dataset, which are subsequently clustered. A deep residual network (ResNet) is trained using patches extracted from each SIFT location (keypoint) using the cluster membership as target. The activations of the penultimate layer serve as local feature descriptors that are subsequently encoded and classified.

2.2.2 ICDAR2017-CLAMM

ICDAR2017 Competition on the Classification of Medieval Handwritings in Latin Script

- **Dataset Description** : Gray level images, annotated with a single script class from 12 classes representing different styles of character shapes used by scribes in producing manuscripts in Europe in the years 500 C.E to 1600 C.E[24].
- **Training/Testing size** : 3500/2000 pages.
- **The Winner title** : Convolutional Neural Networks for Font Classification.[25]

- **Accuracy** : 85.20%.
- **Features learning** : Unsupervised.
- **Lazy/Eager learner** : Eager.
- **Summary** : A CNN is trained to classify small image patches into font classes. At prediction time, we densely extract patches from the test image and average font predictions over individual patch predictions.

2.2.3 ICDAR2017-Historical-WI

ICDAR2017 Competition on Historical Document Writer Identification

- **Dataset Description** : In the train set, three gray-level images by writer VS five in the test set all from 13 to 20-century documents[26].
- **Training/Testing size** : 1182/3600 pages.
- **The Winner title** : Writer Identification on Historical Documents Using Oriented Basic Image Features[27].
- **Accuracy** : 72.40%.
- **Features learning** : handcrafted
- **Lazy/Eager learner** : Eager.
- **Summary** : We exploit handwriting texture as the discriminative attribute characterizing the writer of a given document. The textural information in handwriting is captured using a combination of oriented Basic Image Features (oBIFs) at different scales. Classification is carried out using a number of distance metrics which are combined to arrive at a final decision

2.2.4 ICDAR-2019-HDRC-IR

ICDAR2019 Competition on Image Retreival for Historical Handwritten Documents

- **Dataset Description** : Gray level images with different samples per writer, i.e. 1-5 plus the ICDAR2017 data set, the main focus of the corpus is the writers of books in the European Middle Age[28].
- **Training/Testing size** : 5982/20000 pages.
- **The Winner title** : Offline Writer Identification based on the Path Signature Feature [29].
- **Accuracy** : 92.50%.
- **Feature learning** : handcrafted.
- **Lazy/Eager learner** : Eager.
- **Summary** : Sift descriptors and pathlet features are computed and projected into a subspace of lower dimensionality and then concatenated. The projection matrices were obtained by performing SVD on the feature matrices of the training dataset. The concatenated vectors were further power-normalized and l2-normalized and used as the final feature vectors.

2.3 Need more info ?

We dont try here to provide a survey of all the methods in the field of writer identification, if that seems unfortunate to you, we recommend [30] and [31], and for a survey of existing databases we recommend [32].

Chapter 3

Pipeline design for writer identification

3.1 General overview

As stated in the previous chapters, we will restrict ourselves to offline text-independent automatic writer identification based on Latin handwritten text.

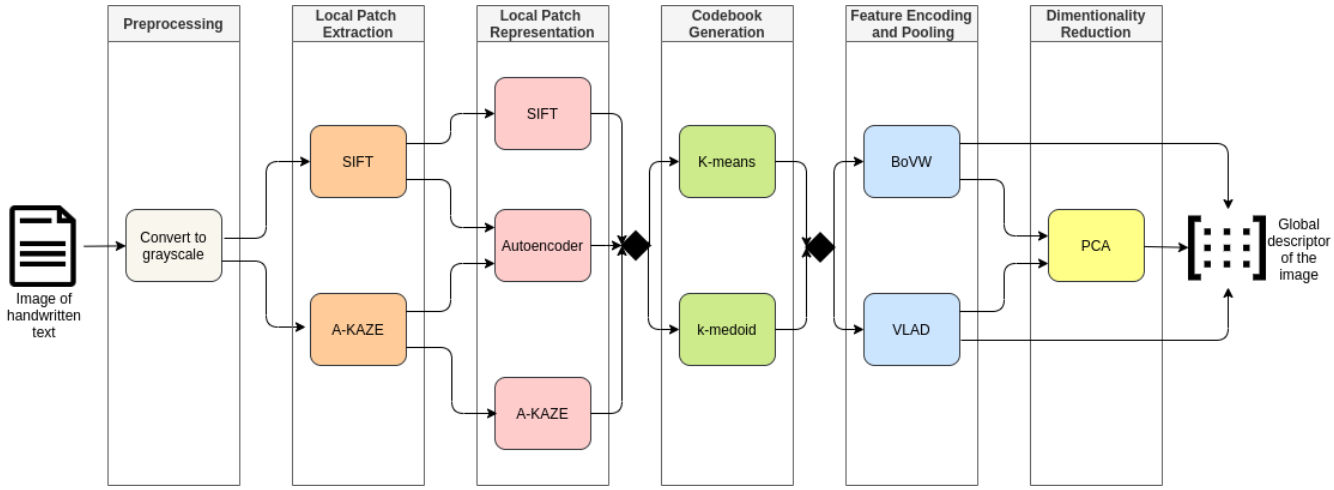


Figure 3.1: Pipeline for our global descriptor construction

In all approaches, we compute the descriptors of the key-points, generate a codebook based on training data, encode the descriptors using statistical methods, and finally extract the principal components of the encoding vector.

3.2 Preprocessing

Ok, how to say it without provoking a computer scientist?
Apart from converting the images to grayscale, no further preprocessing is needed:

- The databases that we will use are clean enough (no pictures of cats nor uniform geometrical figures in the background),
- Using morphological operators risk creating some local patterns which will impact the accuracy of the models (hypotheses tested),
- The methods chosen for key points extraction are robust faced by affine operations and partially invariant to illumination changes.

3.3 Local Patch Extraction

3.3.1 Overview

At this step, we would like to detect all the points of the images (2D) that stand out, where the surrounding can be re-found in other images, where the intensity is not constant with respect to any certain direction. On a white page: None, but in a picture of handwritten text all crossing corners, end of line edges, special dots, and intersection points mist be detected. To top it, they must be invariant to uniform scaling, orientation, and illumination changes.

Formally, what we are searching for is called an interest point, a point in the image which in general can be characterized as follows[33]:

- have a clear, preferably mathematically well-founded, definition,
- have a well-defined position in image space,
- have local image structures around the interest point that are rich in information content such that the interest points carry important information to later stages,
- be stable under local and global deformations of the image domain, including perspective image deformations and illumination variations such that the interest points can be reliably computed with a high degree of repeatability,
- be sufficiently distinct, such that interest points corresponding to physically different points can be kept separate.

Different algorithms are developed and implemented for such purpose; in the following parts, we will present the ones that we will be using: SIFT and A-KAZE.

3.3.2 SIFT

The scale-invariant feature transform algorithm, is by far, the most used and studied one that can detect and describe key points.

Patented by the University of British Columbia[34], it was published by David G. Lowe in 1999[35], and later perfectionned at 2004[36].

Before SIFT, most object recognition algorithms were based on Harris corner detection. Being invariant to rotation, but not to scale makes it very sensible (considering that what can be recognized as a corner at a certain scale is just a flat line in another) and not a very good choice for our use case, even if it is computationally better than SIFT.

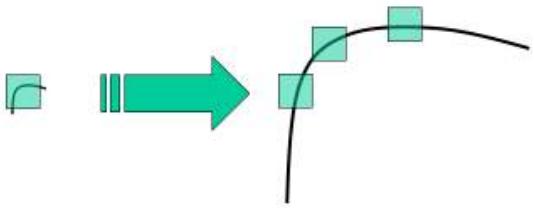


Figure 3.2: Scale invariance[37]

It detects key points based on the local image gradient directions and magnitude at different scales and different values of the variance parameter.

A simple to follow explanation can be found here about how it works: [38].

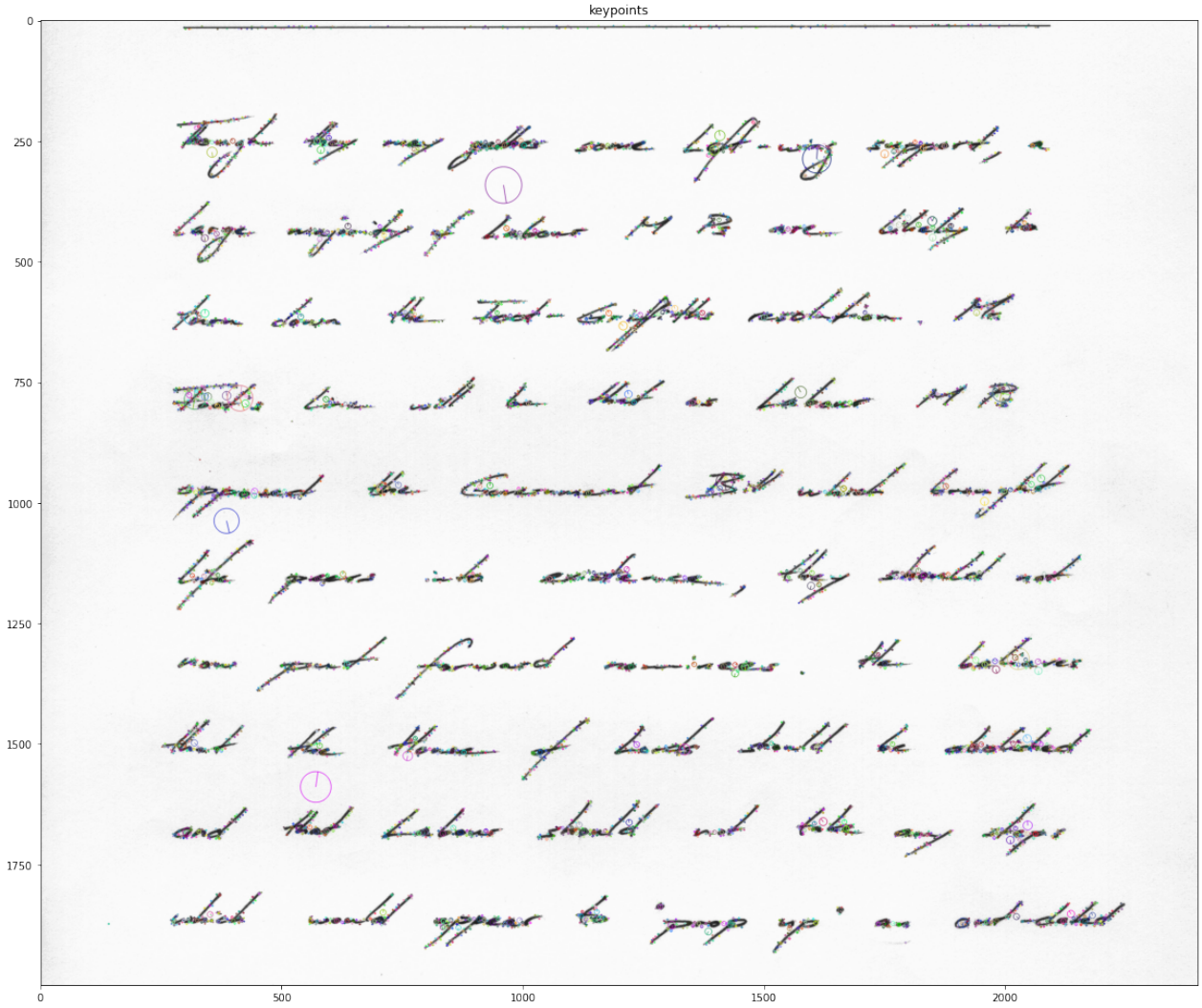


Figure 3.3: 7114 key points detected using SIFT algorithm on one record from IAM database.

3.3.3 A-KAZE

The Gaussian blurring used in SIFT does not respect the natural boundaries of objects since image details and noise are smoothed to the same degree at all scale levels.

To surpass such a problem KAZE algorithm was developed in 2012[39]; its name comes from the Japanese word KAZE that means wind making an analogy with the nonlinear diffusion processes in the image domain (just like the wind in the physical world).

Figure 3.4: Blur with Difference of Gaussian (left), Blur with non-linear diffusion filtering (right) [39]

KAZE algorithm requires solving a series of Partial Differential Equations, done using a numerical method, it is computationally costly, and therefore an accelerated version of KAZE was created. This version is called AKAZE (Accelerated KAZE)[40].

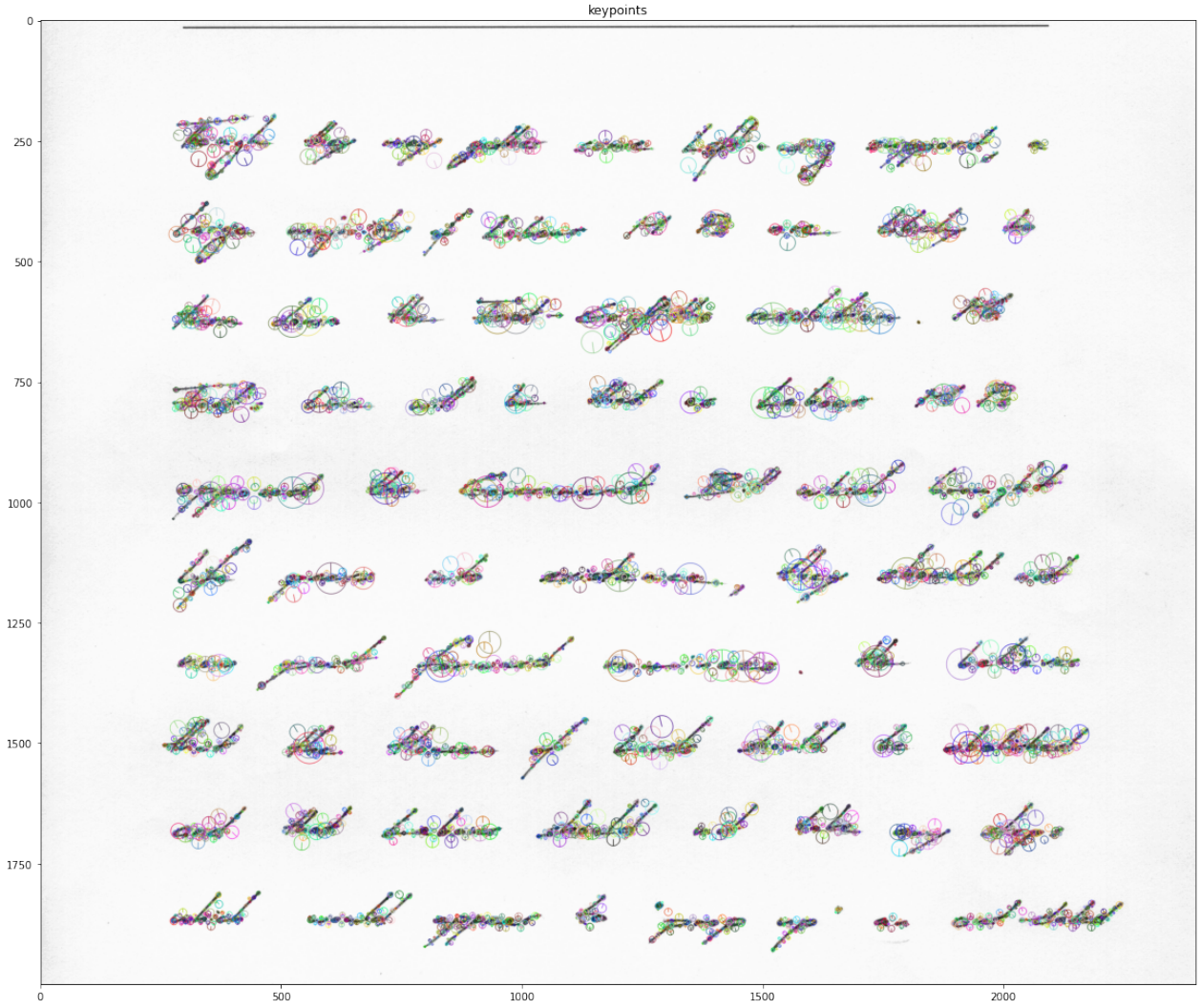


Figure 3.5: 8298 key points detected using A-Kaze algorithm on one record from IAM database.

3.4 Local Patch Representation

For each image, a pool of N image patches is extracted using the key points from the previous step.

Both SIFT and A-KAZE -handcrafted by experts in computer vision-, have their algorithm for generating descriptor vectors: robust to affine image transformations, blurring, illumination, and viewpoint changes there seems like we have nothing to complain about, but as non-experts in computer vision, we only hope to somehow create them our selves, and there no better to do it that using the magical box: Neural Networks.

3.4.1 Autoencoders

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$ [41].

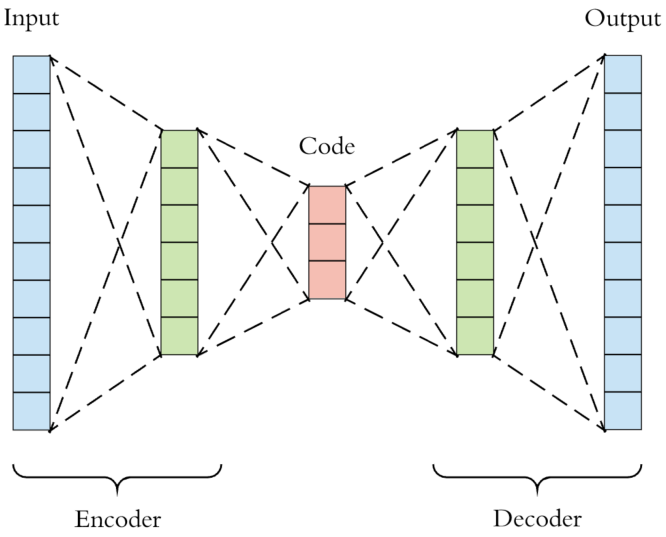


Figure 3.6: Autoencoder general architecture[42]

Seems stupid (I mean, why would any sane person use a deep learning algorithm for something as simple as returning the input), but actually it is one of most user architectures considering its application:

- Image Denoising,
- Recommendation system,
- Anomaly Detection,
- Machine Translation,
- Dimensionality Reduction...

We won't go into details of how it could be employed in each use case apart from ours: dimensionality reduction.

Successfully applied many times for face encoding [43] and less times for writer identification [44] the principle stay the same, even the configuration/architecture we are using was developed for the presented for noise reduction in the MNIST database (of course we did some minimal changes to the hyperparameters, more on that in the implementation chapter).

We train an autoencoder to learn characteristics from the data by encoding them into a lower sized layer, which is then decoded to try to replicate the input. The feature vectors are obtained by forwarding the patches through the trained encoder, taking information from the intermediate layer of an autoencoder.

3.5 Codebook Generation

3.5.1 Purpose

In the same way that to write we use a finite set of letters, to localize a point we use a localization base, to characterize texture we need a finite set of local features that could be used later to describe the distribution of the bigger set of local features in an image.

What we are talking about is called a codebook(i.e., a texton dictionary) that we will be learning using k-mean from the training data.

3.5.2 K-means clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster[45].

Lloyds algorithm

This is the simplest algorithm to solve it, and by no mean the version that you and we should use, other variations use better heuristics for initialization, treat special cases and outliers, and are computationally better[46].

Given a training set $x_1, ..., x_m$ where $x_i \in R^n$, we group the data into k cohesive clusters. Our goal is to predict the centroids of the clusters and a label c_i for each data-point with the minimum within-cluster variance (based on a given convergence criteria):

Algorithm 1: Lloyds algorithm for K-means clustering

Initialization: k and convergence criteria;
Return: $\{\mu_1, ..., \mu_k\}$;
Initialize cluster centriods randomly;
while *Stopping condition not satisfied* **do**
  foreach $i \in \{1, ..., m\}$ **do**
    $c_i := \text{argmin}_{j \in \{1,...,k\}} \parallel x_i - \mu_j \parallel_2^2$
  end
  foreach $j \in \{1, ..., m\}$ **do**
    $\mu_j := \frac{\sum_{i=1}^{m} 1\{c^i = j\} x^i}{\sum_{i=1}^{m} 1\{c^i = j\}}$
  end
end

The size of the codebook is a hyperparameter that we will choose later empirically.

Lloyds algorithm stopping conditions

- Variance less than a particular threshold,
- Maximum number of iterations attained,
- Variance did not improve by at least a certain value,
- Variance did not improve by at least a certain value times the initial variance.

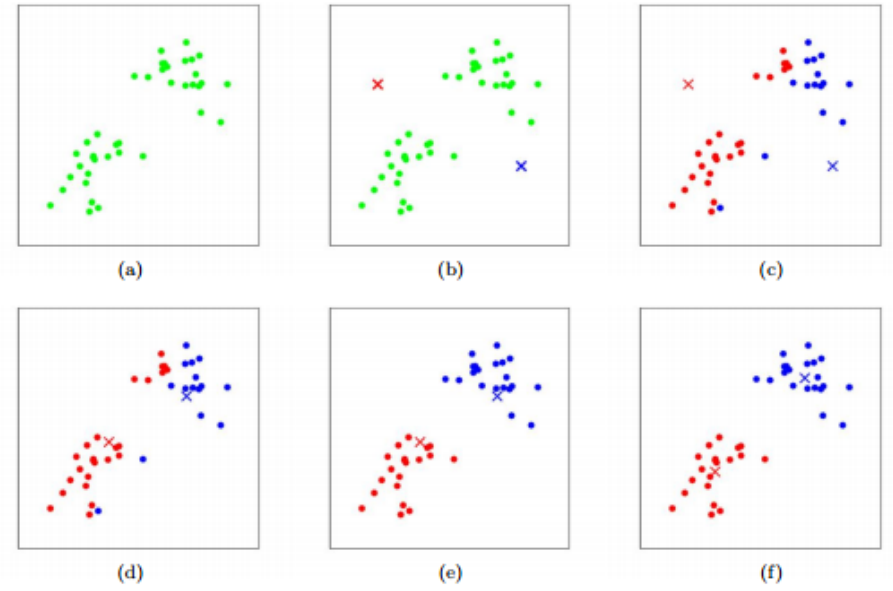


Figure 3.7: K-means algorithm. Training examples are shown as dots, and cluster centroids are shown as crosses. (a) Original dataset. (b) Random initial cluster centroids. (c-f) Illustration of running two iterations of k-means. In each iteration, we assign each training example to the closest cluster centroid (shown by "painting" the training examples the same color as the cluster centroid to which is assigned); then we move each cluster centroid to the mean of the points assigned to it[47]

.

3.5.3 Mini-batch k-means

First presented in 2010[48] and already supported by many machine learning libraries.

Instead of using the full dataset at each iteration, Mini-batch k-means is (as the name suggests) a k-means variation that uses a new random sub-sample of the data points at each iteration, therefore it has less computational complexity: Instead of calculating the Euclidean distances on all dataset, it computes it on a smaller sample, and update the coordinates of the centroids based on the sub-sample (doing it enough iterations and it will converge to a near-optimal value).

Apart from the computational benefit, mini-batch k-means used on large datasets is more robust faced with local minima (sub-samples are random, each with its minima).

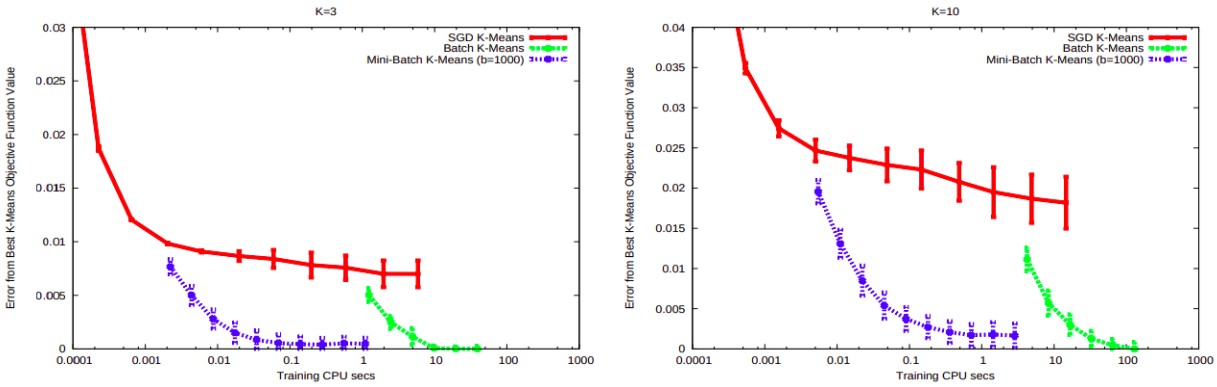


Figure 3.8: Convergence Speed. The mini-batch method (blue) is orders of magnitude faster than the full batch method (green)[48].

3.5.4 k-medoids clustering

The k-medoids or partitioning around medoids (PAM) algorithm is a clustering algorithm reminiscent of the k-means algorithm. Both the k-means and k-medoids algorithms are partitional (break the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses data points as centers and can be used with arbitrary distances, while in k-means the centre of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster)[49].

Generally, k-medoids classification is more robust and flexible than k-means (not picky about the distance metric), it's mean drawback is that it's more expensive computationally: $\mathcal{O}(n^2ki)$ against $\mathcal{O}(nki)$ for the k-means (using the common implementation)[50].

3.6 Feature Encoding and pooling

Given the generated codebook and the extracted local features $\{x_i\}$ from the image, we represent each local feature x_i with the codebook before aggregating them, to finally obtain a global descriptor for the entire image.

The models that we will be using are BoVW and VLAD.

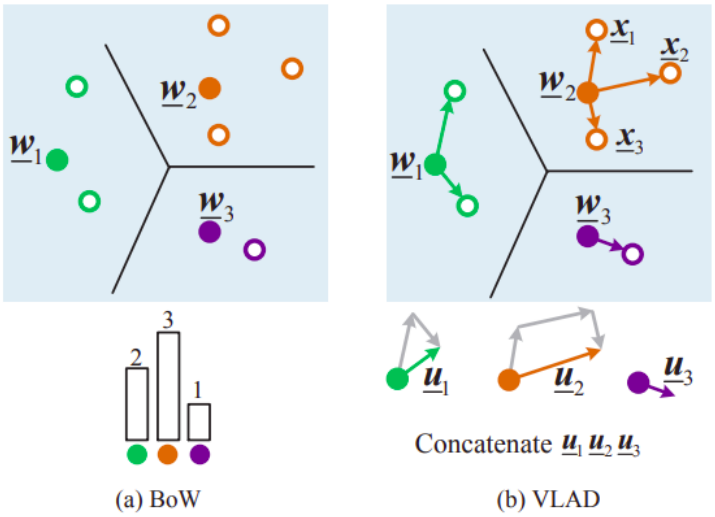


Figure 3.9: Contrasting the ideas of BoVW and VLAD.(a) BoVW: Counting the number of local features assigned to each codeword. It encodes the zero order statistics of the distribution of local descriptors. (b) VLAD: Accumulating the differences of local features assigned to each codeword.[17]

3.6.1 BoVW

By far, the simplest model is the bag-of-visual-words.

Given a set of codewords (local texture descriptors) $\{x_i\}, i \in \{1, ..., n\}$ (n is the number of codewords), we assign each one to one cluster (hard voting) based on the codebook $\{\mu_i\}, i \in \{1, ..., k\}$ (k is the number of clusters of the previous step:

$$v(i) = \begin{cases} 1 & \text{, if } i = \text{argmin}_j(\| x - \mu_j \|) \\ 0 & \text{, otherwise.} \end{cases} \tag{3.1}$$

After that, we do sum pooling (fancy expression that means do the sum of all vectors) which give us a histogram (just a 1D-vector), with 1*(size of codebook) size that we finally normalize using l1 norm (l2 will introduce more sparsity in the vector which we really don't need).

3.6.2 VLAD

Vector of locally aggregated descriptors[51], known as a more powerful representation than BOW[52] and a simplification of the Fisher kernel.

In the same way as for BoVW, we make use of the codebook$\{\mu_1, ..., \mu_k\}$ (that we created in the previous section where k is the number of clusters), but instead of just counting the frequency of local features assigned to each centroid we accumulate the residuals:

$$v_i = \sum_{x \text{ such that } NN(x)=\mu_i} x - \mu_i \tag{3.2}$$

Then concatenate all of them : $v := (v_1^T, ..., v_k^T)^T$ which results in a matrix of size k*n (where n is the size of one local feature).

Finally, we do intra-normalization (each vector v_i is l2 normalized individually), and l2 normalizes the full matrix.

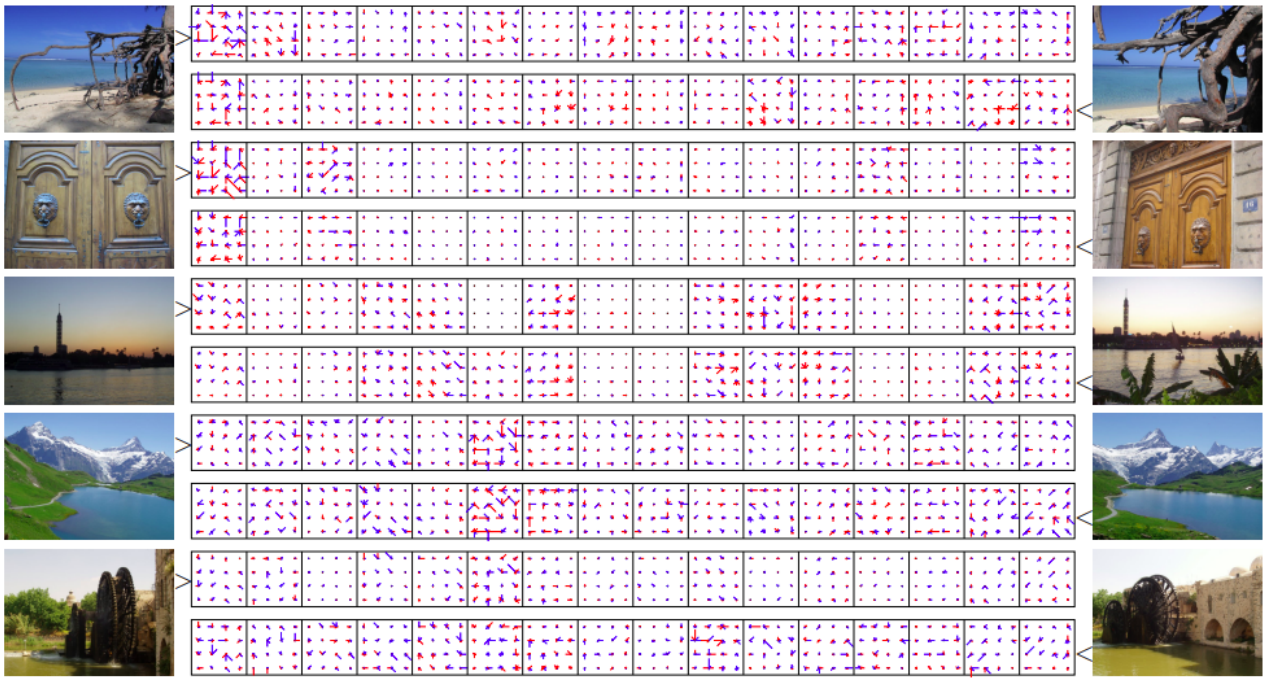


Figure 3.10: Images and corresponding VLAD descriptors, for k=16 centroids (D=16Œ128). The components of the descriptor are represented like SIFT, with negative components.[51]

3.7 Dimentionality reduction: PCA

At this point we already have a global descriptor that represents the texture of the entire image, why reduce it?

Let's take the example of sift algorithm with VLAD, the resulting vector will be of size :

N = (sift descriptor size)*(number of clusters chosen for kmeans)

The factor on the right is 128 (constant), the right one is a hyperparameter (no fixed value as stated earlier), let's just use 350 as a value (later we will explain how to get a good approximation that increase accuracy), the result is:

N = 44800.

Do you see the problem? it's a big matrix, the bigger it is the more computational resources are needed (resources=money).

Luckily we already discussed a method for dimensionality reduction: using autoencoders, but for the sake of diversification (satisfying one of the objectives), we will be using a different model: Principal component analysis technique.

The idea is to extract the eigenvectors of the data's covariance matrix using single value decomposition factorization, normalize the eigenvectors making them unit vectors that will transform the covariance matrix into a diagonalized form with the diagonal elements representing the variance of each axis.

The purpose is to keep the components with the highest variance (how much to keep a hyperparameter, luckily we have an entire chapter to determine all of them).

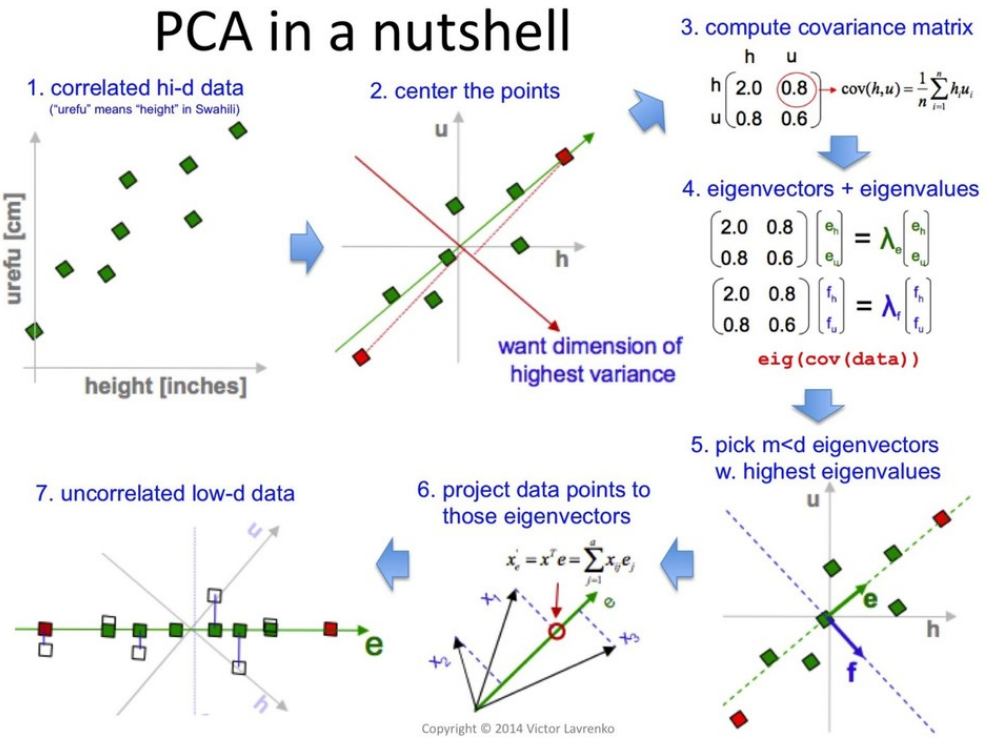


Figure 3.11: PCA resume.[53]

3.8 Similarity Measures

Finally, to determine the writer, we use distance-based classification: the descriptor of the query document is compared to the each of the global descriptors of the references documents (do you see now why we have interest in reducing the size of the global descriptor vector), which return a ranked list of similarity where the writer of the top vector/documents the writer of the query: exactly like the k-nearest neighbors algorithm with $k = 1$.

Faced with the multitude of distance measures: euclidean, manhattan, hamming, chi-squared, cosine ... we referred to previous researches and choose the following three :

- Chi-squared distance[54] : A true metric, defined as :

$$\chi^2(x, y) = \frac{1}{2} \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i} \tag{3.3}$$

  We will be using it with SIFT and A-KAZE.

- Cosine distance[23] : Defined by the cosine of the angle between two vectors (of course of the same dimensionality) :

$$\text{cosineDistance} = 1 - \cos(\theta) = 1 - \frac{x.y}{\| x \| \| y \|} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3.4}$$

  regrettably, it is not a metric (doesn't satisfy the triangle inequality).

3.9 Data structures for reducing search time complexity

A simple brute force method with a computational complexity of $\mathcal{O}(n^2)$ is fine while we are handling a small set of data points, which is not the case in the testing step, so we need better approaches: tree-based data structures that take advantage of the properties of true distances to come up with good heuristics (good in the sense of faster).

As stated, the only requirement is a distance function that satisfies the properties of a metric space, as we saw in the previous section that's not always a given, so not all encoding algorithms will benefit from this section.

3.9.1 Ball tree data structure

Disclaimer: This section have been previously published in [55].

A ball tree is a binary tree in which points are organized based on some metric defined on pairs of points. In their original form, each nodes points are assigned to the closest center of the nodes two children. The children are chosen to have maximum distance between them, typically using the following construction at each level of the tree. First, the centroid of the points is located, and the point with the greatest distance from this centroid is chosen as the center of the first child. Then, the second childs center is chosen to be the point farthest from the first one.

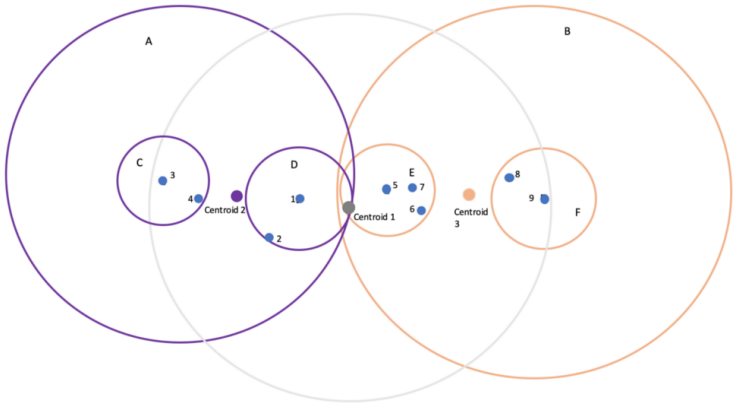


Figure 3.12: Visualization of the Ball Tree Algorithm.

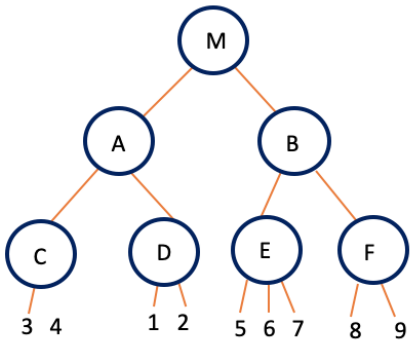


Figure 3.13: The resulting Ball tree. [56]

3.9.2 Vantage Point Trees

Rather than partitioning points on the basis of relative distance from multiple centers (as was the case with ball trees), the vp-tree splits points using the absolute distance from a single center.

Each node of the tree contains one of the input points and a radius. Under the left child are all points that are closer to the node's point than the radius. The other child contains all of the points which are farther away. The tree requires no other knowledge about the items in it[57].

Chapter 4

Implementation of the writer identification pipeline

4.1 Software environment

4.1.1 Programming Language choice

Figure 4.1: Python's Logo

Surprised? it's the obvious choice:

- **Ecosystem support**: It has many libraries for any non-niche algorithm, reducing the need to reinvent the wheel (more on that in the next section), plus its vast community,
- **Previous experiences**: I have already done mini-projects using it, it's not a determinant factor, but it helps,
- **Development time**: Compared with languages from the C family, python is generally faster to get something done (dynamic typing, interpreted, well designed standard library ...)
- **Performance**: yes I admit that python is very slow compared to other languages, but the majority of the heavy lifting is done by libraries that are written in C++, C, or Cython and offer interfaces in python.

4.1.2 Libraries

In this section, We will be presenting some of the most used libraries in our project, for more details, refer to the projects repositories or official documentation, and for a detailed list of the libraries used with the version number, refer to the *requirements.txt* file.

NumPy

Figure 4.2: Numpy's Logo

NumPy is a library that combines the expressive power of array programming, the performance of C, and the readability, usability and versatility of Python in a mature, well tested, well documented and community-developed library.

The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are stridden views on memory that are densely packed arrays of homogeneous type. Python lists, by contrast, are arrays of pointers to objects, even when all of them are of the same type. So, you get the benefits of locality of reference.

Also, many Numpy operations are implemented in C, avoiding the general cost of loops in Python, pointer indirection and per-element dynamic type checking[58].

Scikit-learn

Figure 4.3: Scikit-learn's Logo

Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings[59].

Opencv

Figure 4.4: OpenCV's Logo

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes an easy to use interface for common tasks on digital images and a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms that we will be using in this project, like SIFT, A-KAZE.

OpenCV has more than 47 thousand people of user community and an estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups, and by governmental bodies[60].

Keras

Figure 4.5: Keras's Logo

Being the most used deep learning framework among top-5 winning teams on Kaggle, Keras is an open-source neural network library written in Python and capable of running on top of TensorFlow. Designed

to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Conceived to be an interface rather than a standalone machine learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used.

Even if it seems that we are interacting directly with TensorFlow, actually we are using keras's interface, that's because, in 2007, Google's TensorFlow team decided to support Keras in TensorFlow's core library[61].

4.1.3 Jupyter Notebook

Figure 4.6: Jupyter Notebook's Logo

The Jupyter Notebook is an open-source web application that allows to creating and sharing documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default Jupyter Notebook ships with the IPython kernel which is fine in our case.

If you are wondering: "But you won't be deploying those notebooks to the user? so why bother?", first thanks for reading to this point, second the purpose of the notebooks is to simplify hyperparameters and pipeline tuning before reassembling the final combination in python classes and modules, and it is also used for models training (done just by experts).

4.2 Hardware environment

Taking into account the modest performance of my machine ('modest' is a nice way to describe it) and low network speed, we opted for computational resources on the cloud :

4.2.1 Google colaboratory

Figure 4.7: Google colab's Logo

A Jupyter notebook environment that runs in the cloud, it supports Python3 kernel and stores its notebooks on Google Drive.

It includes two plans, one paid (9.99$ per month) and the other free. We will be using the free one (it's better than my local resources).

The hardware characteristics are not constant; they depend on how much I already used and how much demand there is; the following is the result after 12 hours of no use:

- GPU: Tesla P100 PCI-E, 16GB ,
- CPU: Intel(R) Xeon(R) 2.20GHz ,
- RAM: 13 GB ,

- Disk space: 38.10 GB of non-persistent memory (deleted after instance restart) but could be augmented via mounting google drive file storage,
- offers a TPU,
- has a data ceiling of some dozens of gigaoctets (information not public, just from my experience).

4.2.2 Vast.ai Service

Figure 4.8: Vast.ai's Logo

Using Google colab is a huge improvement compared to working on my local machine, but it's still slow for training and testing and the more I use it the fewer resources I get the next time. On the other hand, it's the best we can get "for free", therefore I am forced to invest real money to rent a machine in the cloud.

From the variety of companies renting private virtual instances online with good GPU's, the most famous ones being GCP from google and AWS from Amazon, but for me, I prefer a more "liberal market" where normal independent persons could rent their machines, outbid each other to get the few users that exist (because it is still new, not all machines have a high reliability, no security measures to secure your data from the host, no big corporation is supporting it, user need minimal command-line knowledge...).

To use vast.ai machines we need a separate storage space, an evident choice is google drive (we already use it for colab), but it does not provide a command-line tool to get to the data (we communicate with the rented machines using ssh) nor could be mount to an exterior notebook, therefore we have come up with two workarounds:

- To Transfer the data to Mega storage and using it's command-line tool megatools (stooped working after a while, probably because of a problem in their servers or that they think that I am misusing their service),
- Mount the drive in google colab, lunch an HTTP server within it, create a tunnel to a public URL from the HTTP server through and use withing vast.ai instance (using ssh) curl or wget to download the dataset (I know it has many risks, but I take them considering that the dataset and the code are already public).

Most of the time, we used variants of this machine :

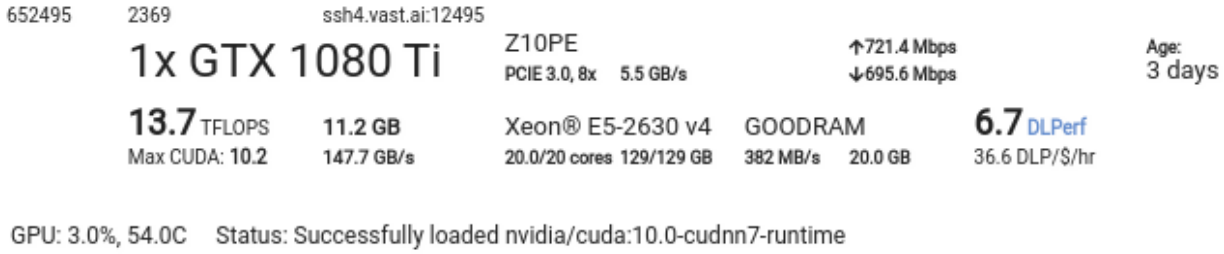


Figure 4.9: Cost less than 0.2 dollars per hour

Note: You are responsible for setting up the machine and you must be aware that the machines aren't 100% reliable.

4.3 The jupyter Notebooks

This is the testing pipline, for the implementation of the classes refer to the source code:

Packages Importation and parameters specification

```
import Clusterer, Global_feature_extractor, Local_features_extractor, Norms, Image, PCA_reduction
import Distances, Autoencoder_train, Encoder_NN
from Dataset_loader import load_dataset
from Accuracy import accuracy_optimised, accuracy
import json, os, cv2, pickle
import numpy as np
from functools import reduce
```

```
PIPELINE_PATHS = {
    "local_patch_extraction": ["SIFT", "A-KAZE"],
    "codebook_generation": ["MiniBatchKMeans", "KMedoids"],
    "feature_encoding_and_pooling": ["BoVW", "VLAD"],
    "dimentionality_reduction": [None, "PCA"]
}
DATASETS = [("IAM", None), ("TrigraphSlant", False), ("TrigraphSlant", True), ("ICDAR", "en"), ("ICDAR", "ar")]
```

```
pipline = [0, 0, 0, 0]

training_session = {
    "id": "Akane",
    "datasets": [0,1,2,3,4],
    "training_size": 7,
    "testing_size": 1
}

if not os.path.exists(training_session["id"]):
  os.mkdir(training_session["id"])
```

```
network_configuration = {
    "shape_images": '?',
    "autoencoder_test_ration" : 0.3,
    "EPOCHS" : 25,
    "BS" :64,
    "latentDim": '?',
    "max_key_points" : 250
}
```

Dataset preparation

```
train_big_set, test_big_set = list(), list()

for choice in training_session["datasets"]:
    train_mini_set, test_mini_set = load_dataset(dataset=DATASETS[choice][0],
                                                 path="./dataset",
                                                 size_train=training_session["training_size"],
                                                 size_test=training_session["testing_size"],
                                                 parametre=DATASETS[choice][1])
    train_big_set.extend(train_mini_set)
    test_big_set.extend(test_mini_set)

_, _, images_train_set = map(list, zip(*train_big_set))
```

```
print("Number of training images:",len(images_train_set))
```

Patchs extraction

```
shapes_images = [8, 16, 32]
```

```
local_patch_extraction_methode = PIPELINE_PATHS["local_patch_extraction"][pipline[0]]

if local_patch_extraction_methode == "SIFT":
    local_features_detector = cv2.xfeatures2d.SIFT_create()
elif local_patch_extraction_methode == "A-KAZE":
    local_features_detector = cv2.AKAZE_create()
```

```
def generate_patchs(folder, local_features_detector, images, shapes_images, max_samples_by_image):
    images_patchs = list()
    shapes_images = sorted(shapes_images, reverse=True)

    retained_patches = list()
```

```
    for i, image in enumerate(images):
        key_points = local_features_detector.detect(image,None)
        retained_patches = list()
        for key_point in key_points:
            retained_mini = []

            y,x = int(key_point.pt[0]),int(key_point.pt[1])
            xm, ym = len(image[0]), len(image)

            max_height = shapes_images[0]
            cropped = image[x-max_height:x+max_height,y-max_height:y+max_height]
            if reduce(lambda x, y: x*y, np.shape(cropped))!=max_height*max_height*4: continue
            retained_mini.append(cropped)

            for shape_image in shapes_images[1:]:
                cropped = image[x-shape_image:x+shape_image,y-shape_image:y+shape_image]
                retained_mini.append(cropped)

            retained_patches.append(retained_mini)

        retained_patches = np.array(retained_patches)
        images_patchs.extend(retained_patches[
            np.random.choice(retained_patches.shape[0],
                             min(max_samples_by_image,len(retained_patches)),
                             replace=False)
        ])
        if i!=0 and i%50==0: print("50 images treated")

    print("Saving the patchs")
    for i, shape_image in enumerate(shapes_images):
        patchs_pickle_path = str(folder)+"/pickle_patchs_"+str(shape_image)+"px.dat"
        with open(patchs_pickle_path, "wb") as f:
            pickle.dump([row[i] for row in images_patchs], f)
```

```
generate_patchs(folder=training_session["id"],
                local_features_detector=local_features_detector,
                images=images_train_set,
                shapes_images=shapes_images,
                max_samples_by_image=network_configuration["max_key_points"])
```

Optimal latent dimention for each patch size

```
latentDims = [8, 16, 32, 64]
```

```
mse_values = list()

for shape_image in shapes_images:
    network_configuration["shape_images"] = shape_image
    patchs_pickle_path = training_session["id"]+"/pickle_patchs_"+str(shape_image)+"px.dat"
    for latentDim in latentDims:
        network_configuration["latentDim"] = latentDim
        model_path = training_session["id"]+"/Encoder_model_"+str(shape_image)+"px_"+str(latentDim)+"elem.h5"

        autoencoder_builder = Autoencoder_train.Autoencoder_train(configuration=network_configuration,
                                                                  data_path=patchs_pickle_path,
                                                                  model_path=model_path)

        mse_values.append(autoencoder_builder.train_network())
```

```
print(mse_values)
```

Packages Importation and parameters specification

```
import Clusterer, Global_feature_extractor, Local_features_extractor, Norms
import Image, PCA_reduction, Distances, Encoder_NN
from Dataset_loader import load_dataset
from Accuracy import accuracy_optimised, accuracy
import os, cv2
import numpy as np
```

```
PIPELINE_PATHS = {
    "local_patch_extraction_representation":[("SIFT", "SIFT"),
                                             ("A-KAZE", "A-KAZE"),
                                             ("SIFT", "Autoencoder"),
                                             ("A-KAZE", "Autoencoder")],
    "codebook_generation": ["MiniBatchKMeans", "KMedoids"],
    "feature_encoding_and_pooling": ["BoVW", "VLAD"],
    "dimentionality_reduction": [None, "PCA"]
}
DATASETS = [("IAM", None), ("TrigraphSlant", False), ("TrigraphSlant", True), ("ICDAR", "en"), ("ICDAR", "ar")]
```

```
pipline = [0, 0, 0, 0]

training_session = {
    "id": "Madoka",
    "datasets": [0],
    "training_size": 10000,
    "testing_size": 10000
}

if not os.path.exists(training_session["id"]):
  os.mkdir(training_session["id"])
```

Dataset preparation

```
train_big_set, test_big_set = list(), list()

for choice in training_session["datasets"]:
    train_mini_set, test_mini_set = load_dataset(dataset=DATASETS[choice][0],
                                                 path="./dataset",
                                                 size_train=training_session["training_size"],
                                                 size_test=training_session["testing_size"],
                                                 parametre=DATASETS[choice][1])
    train_big_set.extend(train_mini_set)
    test_big_set.extend(test_mini_set)

_, _, images_train_set = map(list, zip(*train_big_set))
writers_test_set, images_names_test_set, images_test_set = map(list, zip(*test_big_set))
```

```
print("Number of training images:",len(images_train_set))
print("Number of testing images:",len(images_test_set))
```

Computing local descriptors

```
modules_chosen = PIPELINE_PATHS["local_patch_extraction_representation"][pipline[0]]
```

```
if modules_chosen == ("SIFT", "SIFT"):
  norm = Norms.Norm.hellinger_normalization
  algo = cv2.xfeatures2d.SIFT_create()
  local_features_extractor_descriptor = Local_features_extractor.Local_feature_extractor(algorithm=algo, norm=norm)
elif modules_chosen == ("A-KAZE", "A-KAZE"):
  norm = Norms.Norm.hellinger_normalization
  algo = cv2.AKAZE_create()
  local_features_extractor_descriptor = Local_features_extractor.Local_feature_extractor(algorithm=algo, norm=norm)
else:
    shape_images = '?'
    max_key_points = '?'
    model_path = training_session["id"]+'/?.h5'
    if modules_chosen[0] == "SIFT":
      local_features_detector = cv2.xfeatures2d.SIFT_create()
    elif modules_chosen[0] == "A-KAZE":
      local_features_detector = cv2.AKAZE_create()
    encoder = Encoder_NN.Encoder_NN((shape_images*2, shape_images*2),
                                    max_key_points,
                                    local_features_detector=local_features_detector)
```

```
    encoder.set_model(model_path=model_path)
    local_features_extractor_descriptor = Local_features_extractor.Local_feature_extractor(algorithm=encoder)
```

```
def get_descriptors(local_features_extractor_descriptor, images_train_set, mini_size_sample=550):
    images_pre_clustering = [Image.Image(image, local_feature_extractor=local_features_extractor_descriptor)
                             for image
                             in images_train_set]
    list_local_descriptors = []
    list_local_descriptors_all = []
    for image in images_pre_clustering:
        mini_list_local_descriptors = np.array(image.local_descriptors)
        #repeated two times to ensure that each image can offer the mini_size sample
        list_local_descriptors_all.extend(
            mini_list_local_descriptors[
                np.random.choice(
                    mini_list_local_descriptors.shape[0],
                    len(mini_list_local_descriptors),
                    replace=False)
            ]
        )
        list_local_descriptors.extend(
            mini_list_local_descriptors[
                np.random.choice(
                    mini_list_local_descriptors.shape[0],
                    min(mini_size_sample,len(mini_list_local_descriptors)),
                    replace=False)
            ]
        )
    return list_local_descriptors_all, list_local_descriptors
```

```
descriptors_all, descriptors_sample = get_descriptors(local_features_extractor_descriptor, images_train_set)
print(len(descriptors_sample))
print(len(descriptors_all))
```

Searchig for the optimal value of K

```
clustering_algo = PIPELINE_PATHS["codebook_generation"][pipline[1]]
max_no_improvement = 500
test_values=range(2, 400, 25)
```

```
Clusterer.Clusterer.choose_number_clusters_clustering(vectors=descriptors_sample,
                                                      algo=clustering_algo,
                                                      max_no_improvement=max_no_improvement,
                                                      test_values=test_values,
                                                      verbose=1)
```

Compute the accuracy of the system as a function of K

```
local_patch_representation = PIPELINE_PATHS["local_patch_extraction_representation"][pipline[0]][1]
if local_patch_representation=="Autoencoder":
  distance_metric = Distances.Distance.angular_distance
  accuracy_calculator = accuracy
else:
  distance_metric = Distances.Distance.chi2_distance
  accuracy_calculator = accuracy_optimised
```

```
def principal_components(images_pre, pca_model_path, percentage_variance = 0.98):
    global_descriptors = [image.global_descriptor for image in images_pre]
    PCA_reduction.PCA_reduction.plot_variance_nbComponents(
        vectors=global_descriptors,
        percentage_variance=percentage_variance)
    PCA_reduction.PCA_reduction.create_new_pca_model(vectors=global_descriptors,
                                                     path_to_save=pca_model_path,
                                                     percentage_variance=percentage_variance)

    pca_instance = PCA_reduction.PCA_reduction(pca_model_path)
    return pca_instance
```

```
accuracy_values=[]
```

```
set_nb_clusters = []
module_chosen = PIPELINE_PATHS["feature_encoding_and_pooling"][pipline[2]]
```

```
images_pre = [Image.Image(image,image_name=image_name,local_feature_extractor=local_features_extractor_descriptor)
              for image, image_name
              in zip(images_test_set,images_names_test_set)]
```

```
for nb_clusters in set_nb_clusters:
    clusters_centers_path = training_session["id"]+"/Centers_clusters_"+str(nb_clusters)+"nb.npy"
    Clusterer.Clusterer.fit_new_trainig(vectors=descriptors_all,
                                        algo= clustering_algo,
                                        path_to_save=clusters_centers_path,
                                        nb_clusters=nb_clusters,
                                        max_no_improvement=max_no_improvement,
                                        metric=None,
                                        verbose=0)
    if module_chosen == "BoVW":
      global_feature_extractor = Global_feature_extractor.BOW(clusters_centers_path=clusters_centers_path)
    elif module_chosen == "VLAD":
      global_feature_extractor = Global_feature_extractor.VLAD(clusters_centers_path=clusters_centers_path)

    [image.set_global_descriptor(global_feature_extractor) for image in images_pre]

    if (PIPELINE_PATHS["dimentionality_reduction"][pipline[3]] == "PCA") and (module_chosen == "VLAD"):
        percentage_variance = 0.95
        pca_model_path = training_session["id"]+"/pca_model_"+str(nb_clusters)+"clusters.pkl"
        pca_instance = principal_components(images_pre, pca_model_path, percentage_variance)
        global_feature_extractor = Global_feature_extractor.VLAD(clusters_centers_path=clusters_centers_path,
                                                                 pca_instance=pca_instance)
        [image.set_global_descriptor(global_feature_extractor) for image in images_pre]

    accuracy_values.append({nb_clusters:accuracy_calculator(X_test=images_pre,
                                    Y_test=writers_test_set,
                                    distance_metric=distance_metric)
                            })
```

```
print(accuracy_values)
```

Chapter 5

Results and analysis

5.1 Benchmarking databases

For our experiments we will be using:

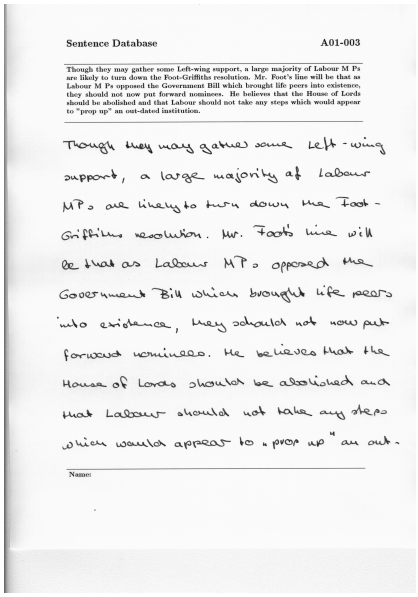


Figure 5.1: IAM example

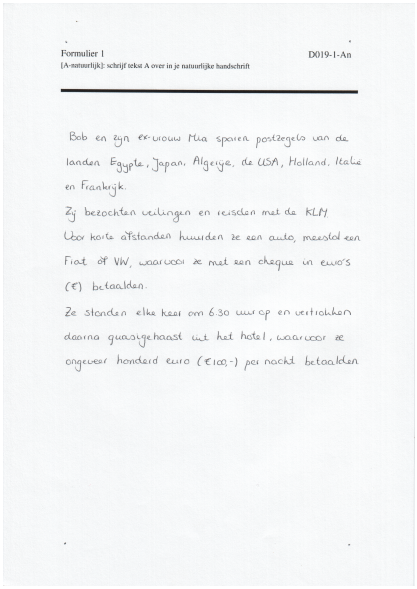


Figure 5.2: TrigraphSlant example

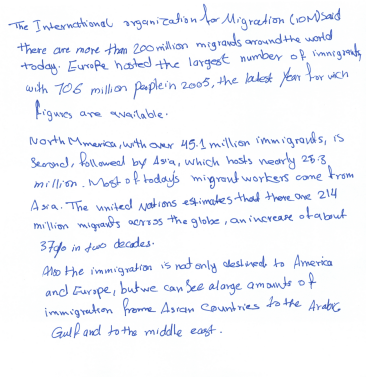


Figure 5.3: ICDAR-GenderIdentifify2013 example

5.1.1 IAM

Publicly available[62], having 931 Citations, the IAM database[63] is widely used in our field of research considering its size, being labeled and segmented (which we don't need in our work but useful for other applications):

- Contains English forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels,
- 657 writers contributed a variable number of handwritten pages,
- 1'539 pages of scanned text,
- 5'685 isolated and labeled sentences,
- 13'353 isolated and labeled text lines,
- 115'320 isolated and labeled words.

5.1.2 TrigraphSlant

Publicly available [64], small compared to the IAM databases, it was created[65] to study the effects of slant on writer identification:

- Consists of 188 scanned images of handwritten pages in Dutch, written by 47 authors (4 pages from each),
- The writers were provided two printed Dutch texts, text A and text B:
  - Copy text A in your natural handwriting.
  - Copy text B in your natural handwriting.
  - Copy text B and slant your handwriting to the left as much as possible.
  - Copy text B and slant your handwriting to the right as much as possible.

5.1.3 ICDAR-GenderIdentify2013

Also Publicly available[66], a subset of the QUWI dataset, it was created[67] to study gender identification based on handwriting:

- Images have been acquired using a professional scanner, with a 300DPI resolution,
- 475 writers produced 4 handwritten documents each :
  - An English handwritten text wich varies from one author to the other.
  - An English handritten text wich is the same for all of the writers.
  - An Arabic handwritten text wich varies from one author to the other.
  - An Arabic handritten text wich is the same for all of the writers.

5.1.4 Train/Test separation

Dataset	Training size	Testing Size
IAM	937	602
TrigraphSlant	32	62
ICDAR2013	300	650

This values holds except when training the convolutional network, then we add the Arabic texts of the ICDAR2013 database and the deformed slant of the TrigraphSlant database:

Dataset	Training size	Testing Size
TrigraphSlant	32	62
ICDAR2013	300	650

The logic behind those intriguing ratios is that for each image thousands of local descriptors are generated: we don't need that much data for training (in the tests we use sub-samples considering the material limitations), for that reason we use one third for training and the rest for tests (even if the other way around is the most common).

As for the "magical numbers" for the IAM dataset, they satisfy the following conditions of the evaluation protocol:

5.2 Evaluation protocol

In each experience, only one document of the same writer as the query is in the dataset, the rest of N-1 data points are all of $\frac{N-1}{2}$ other writers (2 documents for each writer).

For example, in a dataset of size 100, there will be 50 writers each with 2 documents. In the testing stage, we loop on the documents, remove one each time labeling it as the target, and calculate its distance from the rest of the 99 documents.

To measure the accuracy of the system, we will be using the TOP-1 metric:

Only if the more similar document has the same writer as the query document, it counts as a success.

5.3 Hyperparameters Tuning

As stated in a previous chapter (I hope you read it), our pipeline has many hyperparameters that don't have a mathematical formula to determine them, we instead determine them empirically.

5.3.1 Latent space dimension of the autoencoder architecture

As stated before, we will be using a simple convolutional autoencoder: take architecture from [68], add a l_2 normalization layer before the middle layer and adapt the entry to experiments.

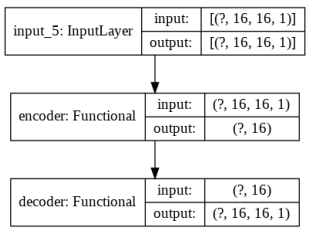


Figure 5.4: Autoencoder

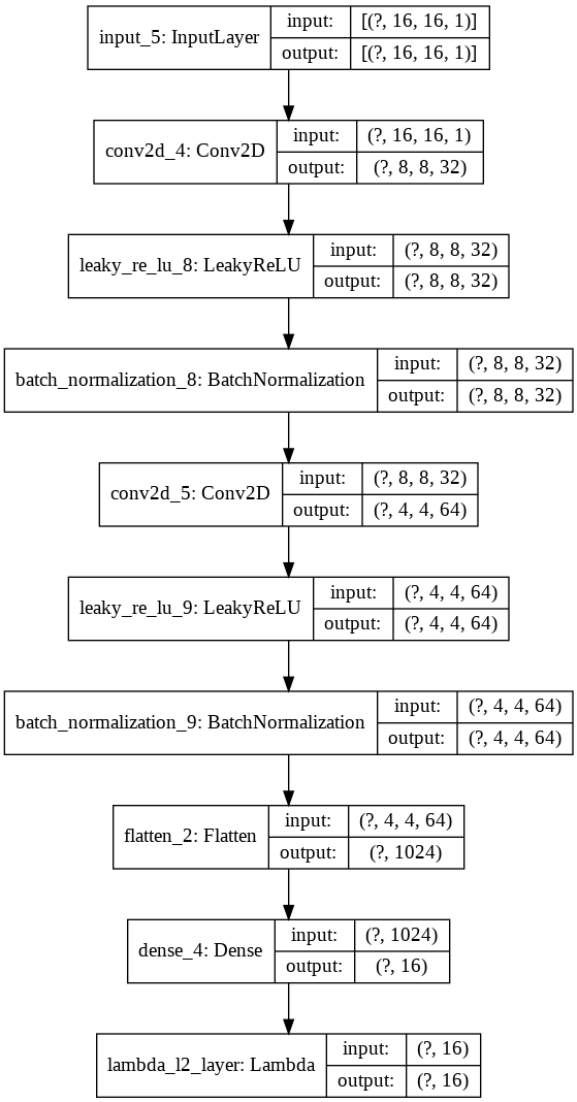


Figure 5.5: Encoder part

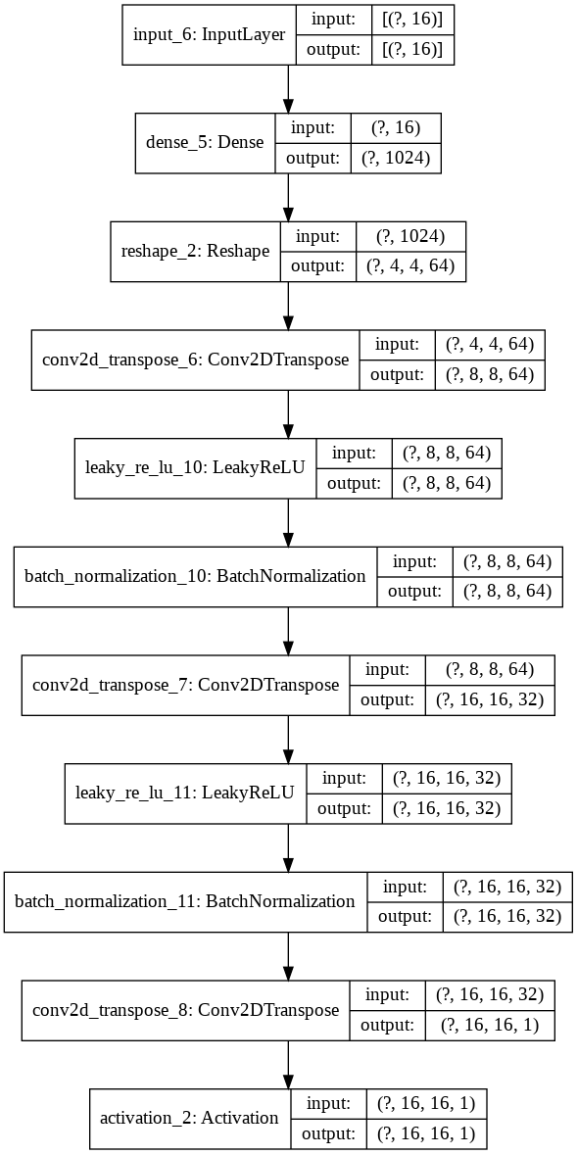


Figure 5.6: Decoder part

As for patch size, we will be testing 16*16, 32*32, and 64*64 patches of 8, 16, 32 and 64 as the latent space dimension; inspect the error and visualize the reconstruction to choose one or more candidates that will make it to the last module; for each of them, we will search for optimal dimension for the latent space.

As for the training data, we will combine the training data of all databases: no benefit in training each one apart (we don't want our model to overfit).

For each image on the training set, I extracted 550 patches surrounding key points detected using the SIFT algorithm (random sampling), resulting in a total of 616096 patches for the training and 264040 for testing.

The reconstruction loss of the autoencoders is calculated using the Mean Squared Error after 25 epochs and 64 as a batch size (for more details on the network refer to the source code):

lattent space dimention\shape	16*16	32*32	64*64
8	0.0078	0.0216	0.0290
16	0.0024	0.0089	0.0180
32	0.0009	0.0031	0.0090
64	0.0004	0.0011	0.0036

As expected the loss of the autoencoder is a decreasing function of the latent space dimension, and as the image gets bigger, it's harder to reconstruct it using the same network architecture:

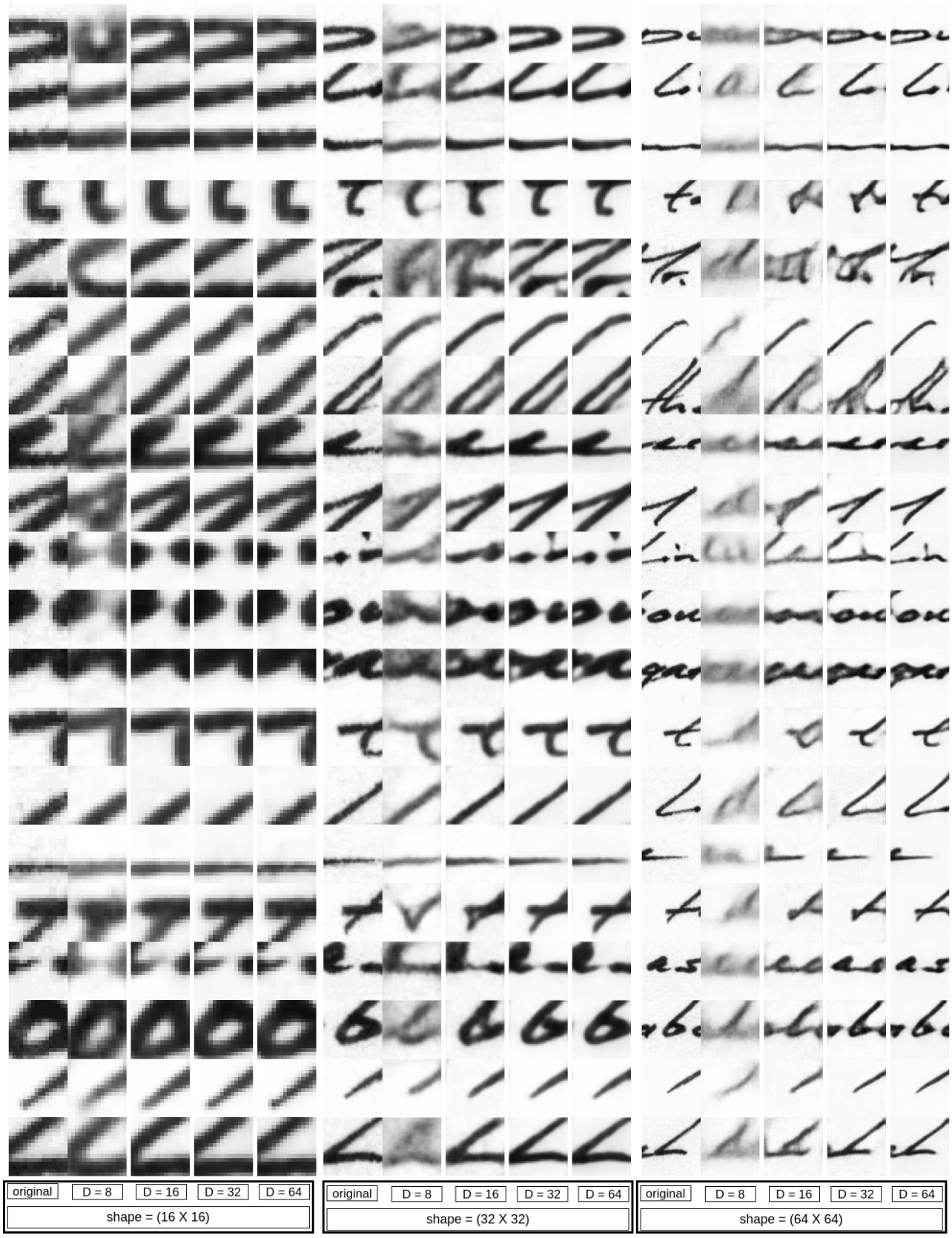


Figure 5.7: Patch reconstruction at different configurations(took 12 hours on the configuration presented in the previous chapter)

5.3.2 Determine the Number of clusters

The most famous heuristic to determine the best number of clusters k for clustering algorithms is the elbow method:

Elbow method

The elbow method is used to determine the optimal number of clusters in k-means clustering. The elbow method plots the value of the cost function (in our case the sum of squared distances) produced by different values of k. As you know, if k increases, average distortion will decrease, each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids. However, the improvements in average distortion will decline as k increases. The value of k at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters[69].

But finding the "elbow" is not always evident:

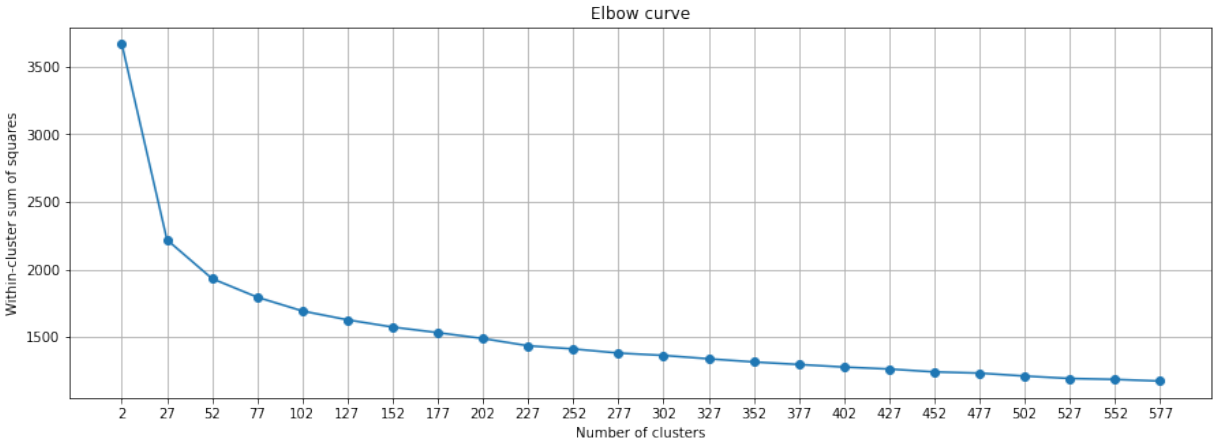


Figure 5.8: WCSS as a function of the number of clusters (applied on IAM training dataset, SIFT=>SIFT=>Mini-batch k-means)

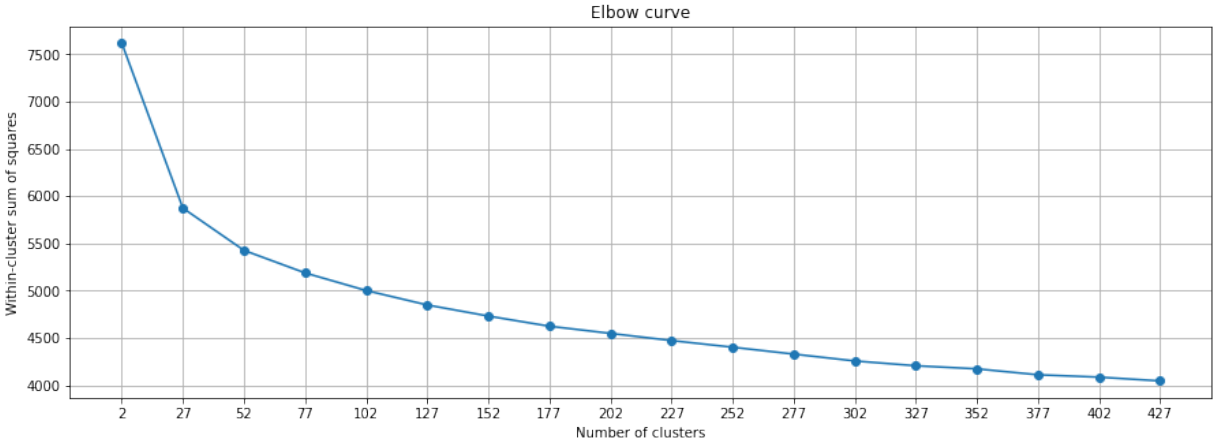


Figure 5.9: WCSS as a function of the number of clusters (applied on IAM training dataset, SIFT=>Autoencoder=>Mini-batch k-means)

Silhouette analysis

The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like the number of clusters visually. This measure has a range of [-1, 1].

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is $\frac{b-a}{max(a,b)}$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is 2 <= n_labels <= n_samples - 1[**?**].

Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar[70].

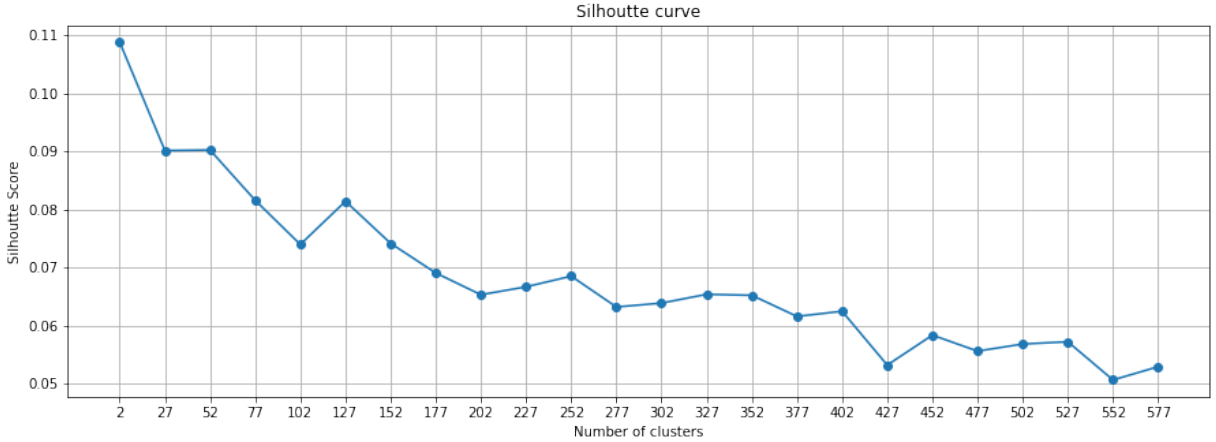


Figure 5.10: silhouette score as a function of the number of clusters (applied on IAM training dataset, SIFT=>Mini-batch k-means)

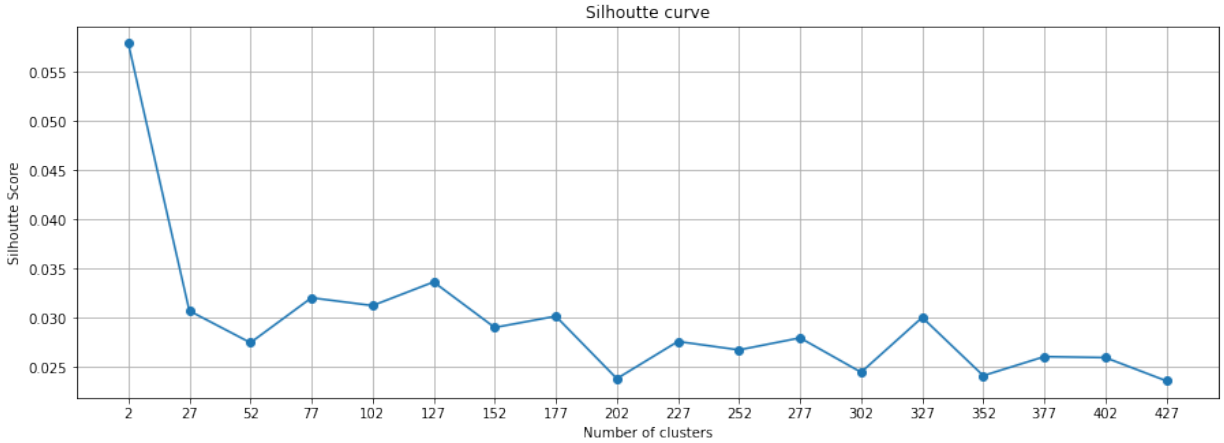


Figure 5.11: silhouette score as a function of the number of clusters (applied on IAM training dataset, SIFT=>Autoencoder=>Mini-batch k-means)

Even this method doesn’t give us an easy to spot visual solution.
Our last result:

Calinski-Harabasz method

Another more image-friendly heuristic is Calinski-Harabasz. Based on capturing the point of change of the mean based on the ration of dispersion between and within cluster.

This time we search for the coordinates of the max when the "withing cluster dispersion" is minimum compared to the "between cluster dispersion":

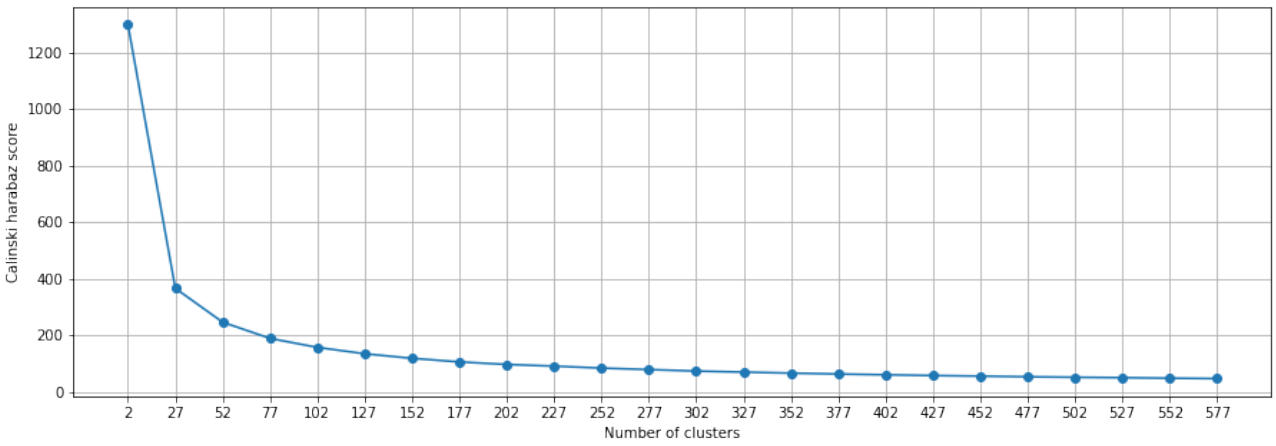


Figure 5.12: Calinski-Harabasz score as a function of the number of clusters (applied on IAM training dataset, SIFT=>Mini-batch k-means)

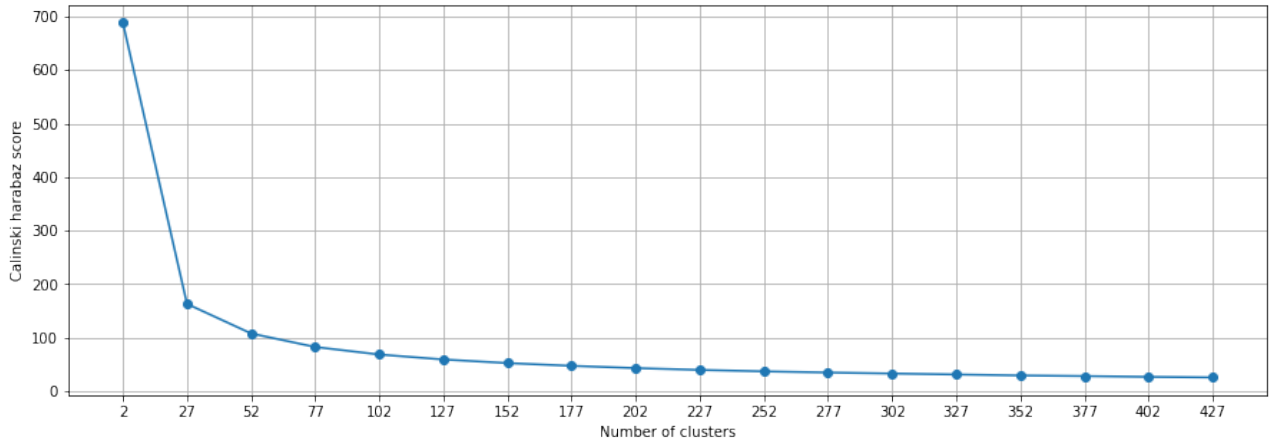


Figure 5.13: Calinski-Harabasz score as a function of the number of clusters (applied on IAM training dataset, SIFT=>Autoencoder=>Mini-batch k-means)

Even after zooming on different intervals, we couldn't get a good enough visual solution (more on this in the last chapter), therefore, We will be using a classical approach:

Accuracy as a function of the number of clusters

I see you there asking, why didn't you just use this approach from the beginning?

Because it's heavily computational, going through all paths to the end, many times to just determine one parameter is considered wasting resources (if there is another method), just like to go from Rabat to Kenitra you don't try all paths before you finally return to Rabat go to Kenitra using "the best path" you "discovered" and claim that you found a scalable solution that could work also on Rabat to Moskow if you have enough resources to spare for a grid search fine, but in the real world that's rarely the case.

We won't show it to you here , wait until the experimentations section.

5.3.3 SIFT vs A-KAZE

On the training set of IAM we found:

- Using SIFT: 2838448 key-points (3029 per document).
- Using A-KAZE: 4667311 key-points (4981 per document).

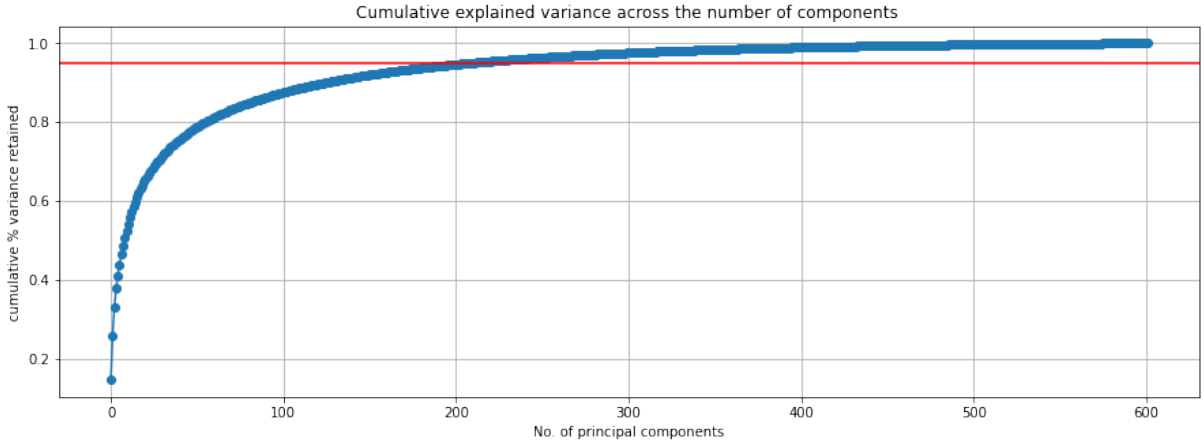
Using the integrated method to compute local descriptors we might find some benefits, but concerning any path using the autoencoder, the bottleneck is the computational power, for this reason, we will use solely SIFT to interact with the autoencoder (which is not bad, taking into account the distribution of key-points on the image and the nature of the dataset: white/pure background).

5.3.4 Number of Principal components

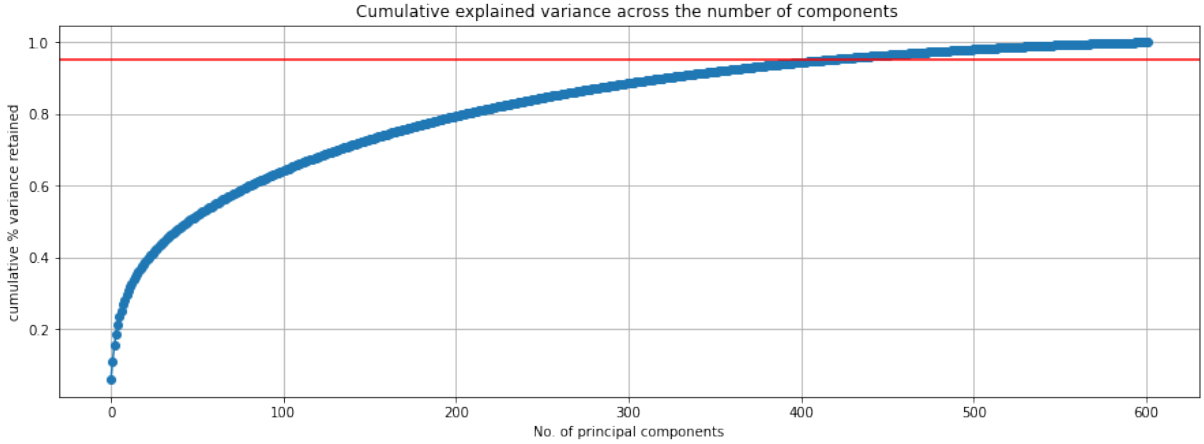
Luckily the implementation of PCA by scikit-librery could select the number of components such that the amount of variance that needs to be explained is greater than a percentage specified.

To get a visualization of what's happening, we draw the cumulative variance as a function of the number of components (the max number of components is the number of minimum of the number of features and samples).

Here it's on the SIFT => Minibatch k-means => VLAD path applied on the IAM dataset (25 clusters) where we kept 95% of the explained variance (from 128*25 to less than 250 components) :



And here applied on SIFT => Encoder=> Minibatch k-means => VLAD (50 clusters):



5.4 Experiments

5.4.1 Accuracy on the IAM dataset

Dataset	IAM						
Local Patch Extraction	SIFT						A-KAZE
Local Patch Generation	SIFT			32*32 Encoder latentDim=64			A-KAZE
Codebook Generation	K-means						
Number Clusters	600	25		350	50		300
Feature encoding and Polling	BoVW	VLAD		BoVW	VLAD		BoVW
Dimentionality reduction (% explained variance)	None		95	None		95	None
% Accuracy	88.87	92,35	89.53	82,39	89,86	88.20	66.61

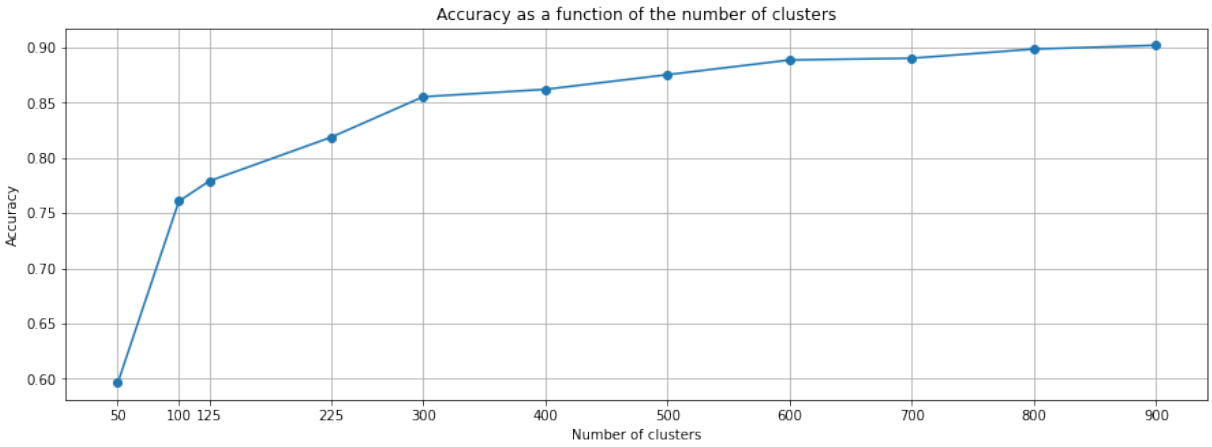


Figure 5.14: IAM database Accuracy using SIFT=>k-means=>BoVW

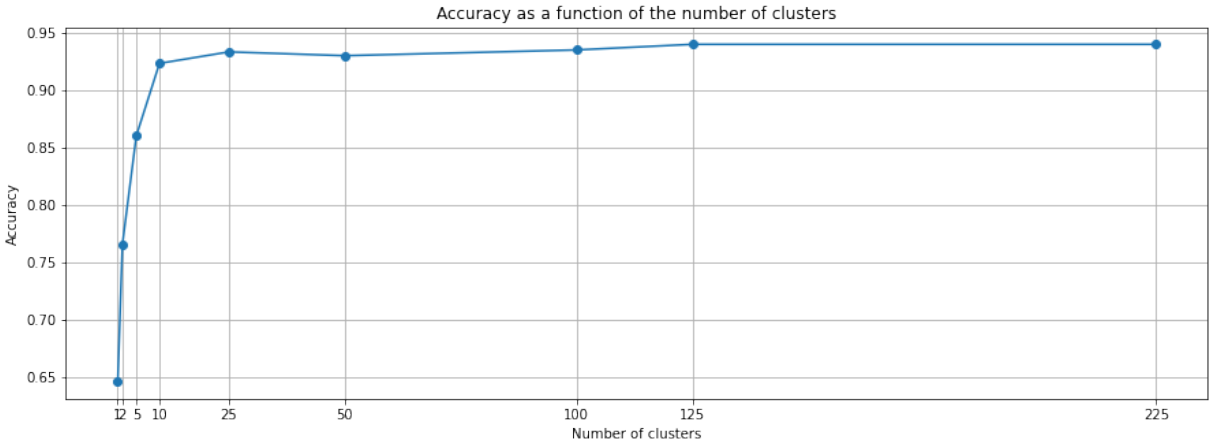


Figure 5.15: IAM database Accuracy using SIFT=>k-means=>VLAD

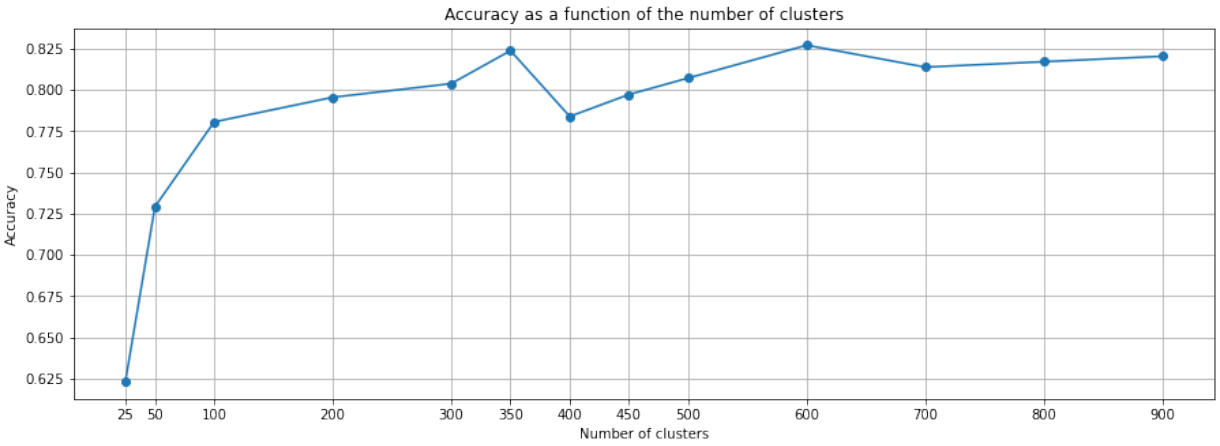


Figure 5.16: IAM database Accuracy using SIFT=>Autoencoder(32*32)=>k-means=>VLAD

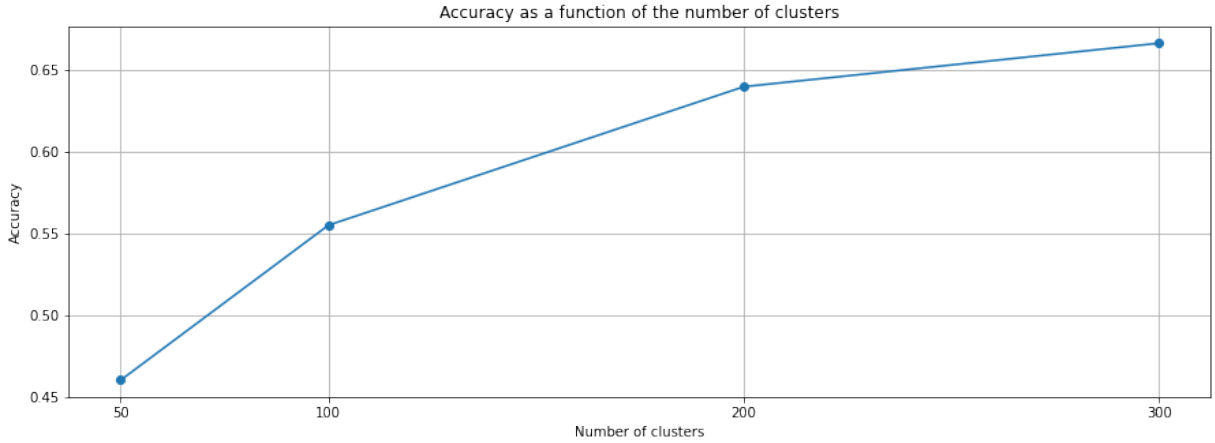


Figure 5.17: IAM database Accuracy using A-KAZE=>k-means=>BoVW

5.4.2 Multi-script identification

Using the path SIFT=>Minibatch k-means => Vlad (25 clusters) we trained a model on the English texts of the ICDAR dataset and obtained 75.38% accuracy, using the same resulting model (that never saw Arabic letters) we tested in on the Arabic texts of ICDAR, the resulting accuracy was 63.07%.

This confirms what we stated earlier in the second chapter:" multi-script identification naturally, have low identification rates".

5.4.3 Impact of the change of slant on the models

Using the path SIFT=>Minibatch k-means => Vlad (25 clusters) we trained a model on the natural handwriting texts of the TrigraphSlant database, the accuracy was 100% (you have all you need to reproduce the result), but testing it on the other part of the dataset (where the slant was deformed), we were surprised that the accuracy was 19%.

The autoencoder path did perform worst: from 71% on normal text to 8%.

We can conclude that our models are sensitive to changes of the slant (at least while not trained on it).

Chapter 6

Machine Learning REST API

As already discussed in the introduction, An important part of this project is to develop a REST API with the following requirements:

6.1 API's Requirements

6.1.1 Functional requirements

- Based on a labeled set of images of handwritten documents and a query image get the label of the most similar document in the reference set,
- Based on a single-labeled set of images of faces and a query picture, get the labels of the faces figuring in the query based on the reference set,
- Based on a single-labeled set of images of faces and a query video, get the labels of the faces figuring in the query based on the reference set,
- Past operations are treated as resources that support: GET, PUT, DELETE,
- Return metrics on the usage of the services.

6.1.2 Non-Functional requirements

- Use JSON as a data-exchange format,
- Improvements in the ML pipeline must be easily transferred to the API,
- Use tokens for authentication and authorization,
- Must be Scaling friendly.

6.1.3 Use case diagram

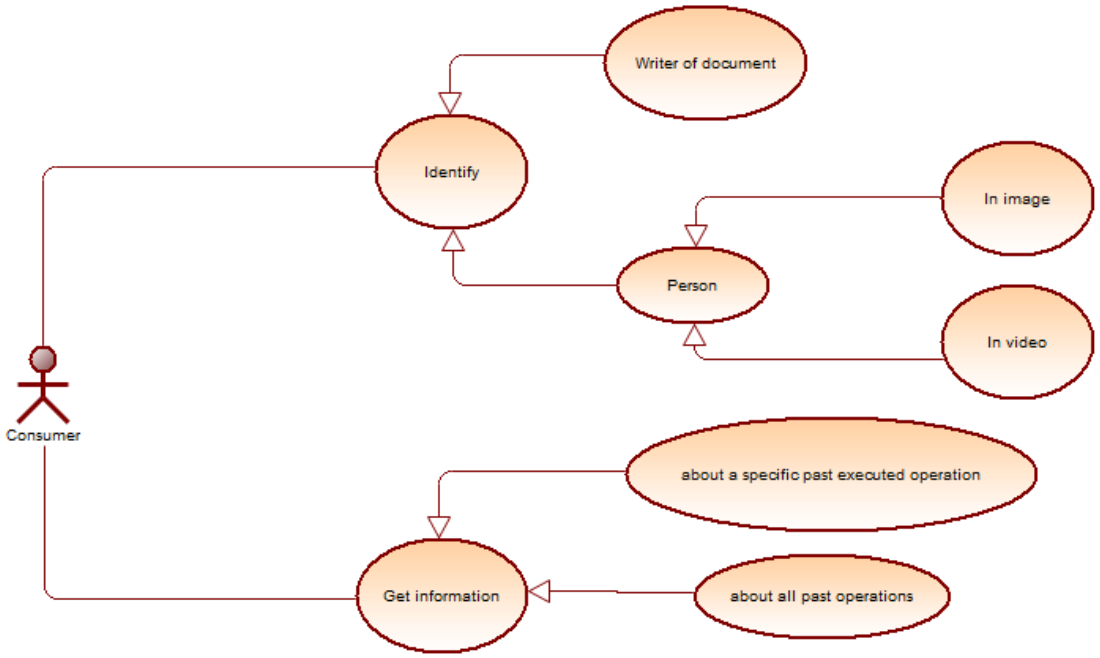


Figure 6.1: Use case diagram

6.2 Design

The general blueprint showing all the endpoints:

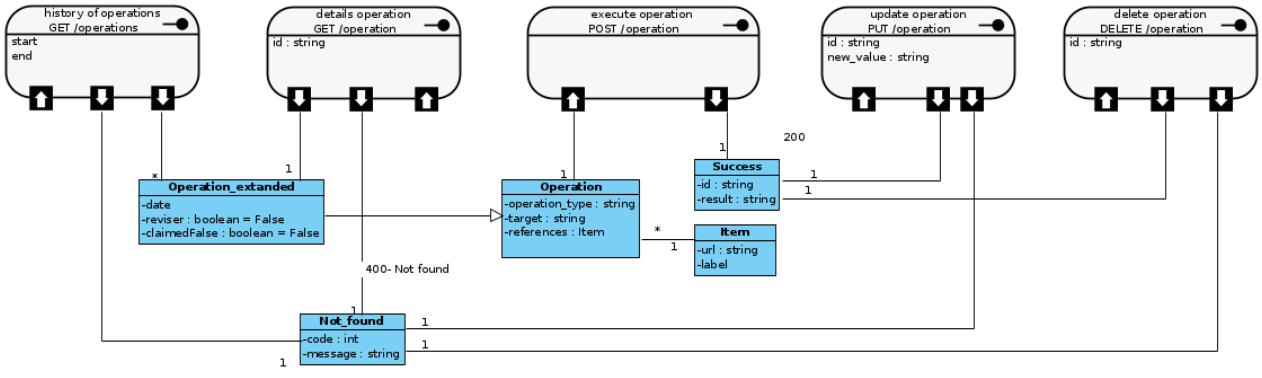


Figure 6.2: Api's blueprint

A big part of the architecture used for the research is kept here (it is reliable enough to be reused); I just added on top of it the face recognition module, data persistence model, and request handlers.

6.2.1 JSON Schema

In this part, I preset the JSON schema for the highly occurring exchanges (no error messages or status codes, for them, refer to the source code).

POST /api/v0/operation/: The client wants to execute an identification operation.

Request:

```
POST BASE_URL/api/v0/operation/
{
    "type": object,
```

```
    "items":{
        "operation_type": {
            "type": "string",
            "enum" : ["identifyFaceInPicture","identifyFaceInVideo","identifyWriter"]
        },
        "references": {
            "type": "array",
            "minItems": 1,
            "items": [
                {"url": {
                    "type": "string",
                    "description": "URL of a reference image"
                    }},
                {"label": {
                    "type": "string",
                    "description": "label of image"
                    }}
                ]
        },
        "target": {
            "type": "string",
            "description": "URL of the query image/video"
        }
    }
}
```

Response:

```
{
    "type": object,
    "items":{
        "id": {
            "type": "string",
            "description": "unique identifier of the operation"
        },
        "results": {
            "type": "array",
            "minItems": 0,
            "items": [
                {
                "type": "string",
                "description": "label of the result"
                }
            ]
        }
    }
}
```

GET /api/v0/operation/: The client wants to get details about a past operation.
Request:

```
GET BASE_URL/api/v0/operation?id={string}
```

Response:

```
{
    "type": object,
    "items":{
        "operation_type": {
            "type": "string",
```

```
            "enum" : ["identifyFaceInPicture","identifyFaceInVideo","identifyWriter"]
        },
        "algo": {
            "type": "string",
            "description": "method used"
        },
        "target": {
            "type": "string",
            "description": "URL of the query image/video"
        },
        "references": {
            "type": "array",
            "minItems": 1,
            "items": [
                {"url": {
                    "type": "string",
                    "description": "URL of a reference image"
                    }},
                {"label": {
                    "type": "string",
                    "description": "label of image"
                    }}
                ]
        },
        "revised": {
            "type": "bool",
            "description": "is the operation revised by a specialist?"
        },
        "claimedFalse": {
            "type": "bool",
            "default": False,
            "description": "did the consumer claim that the result is wrong?"
        },
        "date": {
            "type": "date",
            "description": "date of the operation execution"
        },
        "results": {
            "type": "array",
            "minItems": 0,
            "items": [
                {
                "type": "string",
                "description": "label of the result"
                }
            ]
        },
        "id": {
            "type": "string",
            "description": "unique identifier of the operation"
        }
    }
}
```

PUT /api/v0/operation/: The client wants to update the claim field, stating that the identification operation was a failure or a success.

Request:

```
PUT BASE_URL/api/v0/operation?id={string}&new_value={boolean}
```

Response:

```
{
    "type": object,
    "items":{
        "id": {
            "type": "string",
            "description": "unique identifier of the operation"
        },
        "claimedFalse": {
            "type": "bool",
            "description": "did the customer claim that the result is wrong?"
        }
    }
 }
```

DELETE /api/v0/operation/: The client wants to delete an operation from the record.
Request:

```


DELETE BASE_URL/api/v0/operation?id={string}
```

Response:

```
{
    "type": object,
    "items":{
        "id": {
            "type": "string",
            "description": "unique identifier of the operation"
        },
        "message": {
            "type": "const string",
            "value": "operation deleted"
        }
    }
  }
```

GET /api/v0/operations/: The client wants to get the number of executed operations and claimed false for each day in an interval.
Request:

```
GET BASE_URL/api/v0/operations?start={date,format:"format": "%Y-%m-%d"}&end={date,format:
    "%Y-%m-%d",default:Today}
```

Response:

```
{
    "type": "array",
    "minItems": 0,
    "items": [
        {
        "date": {
            "type": date,
            "format": "%Y-%m-%d"
        },
        "totalSum": {
            "type": integer,
            "description": "number of operation executed at the set day"
```

```
        },
        "totalClaimed": {
            "type": integer,
            "description": "total number of operations claimed false by a user"
        }
    ]
    }
}
```

6.3 Development tools choices

6.3.1 Language and development framework

For the language, we stick with python, apart from the reasons already presented the first time we presented the language (of course you remember, you have read it), we also prefer to reuse here a large chunk of the code already tested and simplify future transferring of the methods from the ML pipeline to the API.

As for the choice of the tools, we will use a micro-framework just for the API. Using going a full-fledged framework is necessary. For this reason, I choose Flask, just pure/clean Flask without big libraries forced on me.

Figure 6.3: Flask's logo

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy[71].

6.3.2 Database Management System

Before announcing the "winner", let's inspect our requirement from the DBS based on the nature of the data:

- We would like to save and fetch the JSON documents with minimum computation,
- The structure of the data is non-rigid; it is based on the clients type of service, and probably will change in the future after chipping the app and collecting feedback from the users,
- Could be outsourced cheaply,
- Has a python driver,
- Big community and good documentation.

Based on all these requirements, we will be using a non-relational database: MongoDB.

Figure 6.4: MongoDB's logo

MongoDB is a document database, which means it stores data in JSON-like documents, supports arrays and nested objects as values, allows for flexible and dynamic schema, and has its proper query language[72].

Also, we will be using MongoDB Atlas: a fully-managed cloud database developed by the same people that build MongoDB. Atlas handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS, Azure, and GCP)[73].

6.3.3 Ngrok

The API was developed while another team was working on a website that consumes my services; to allow for fast testing for the other team, I used:

Figure 6.5: ngrok's logo

ngrok is a reverse proxy that creates a secure tunnel from a public endpoint to a locally running web service it also captures and analyzes all traffic over the tunnel for later inspection and replay.

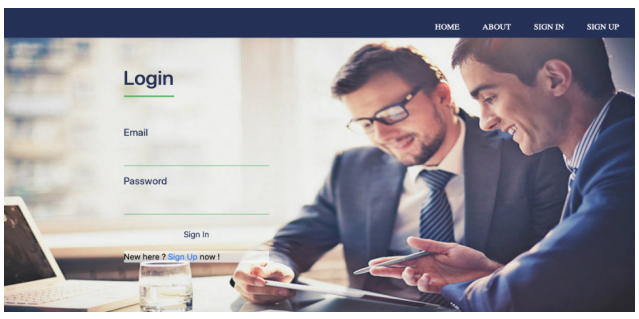
The free plan is largely sufficient:

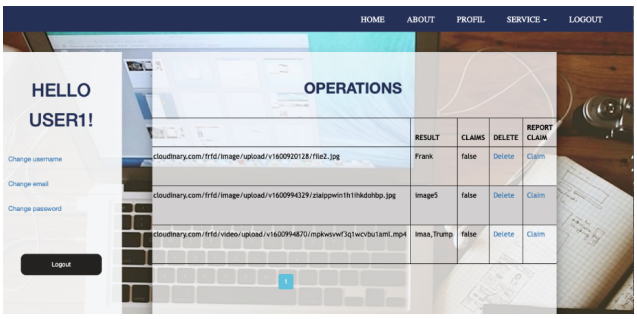
- 1 online ngrok process,
- 4 tunnels / ngrok process
- 40 connections / minute.

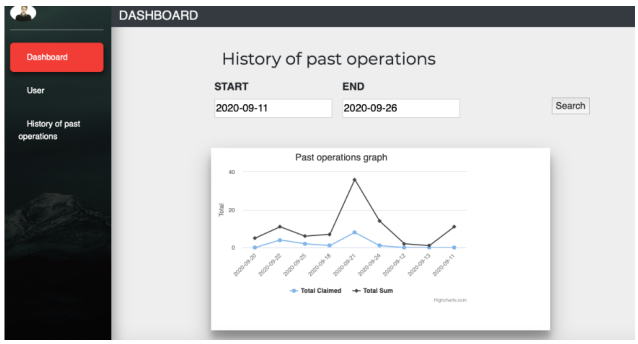
6.4 Why store the data on my end, not on the website end?

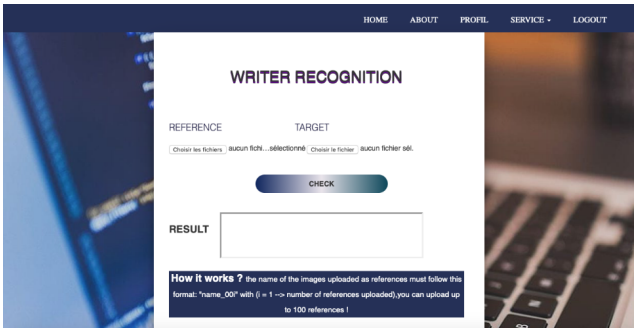
As stated earlier, the development of the API was in parallel with another team working on a website that consumes it, the website in question could manage the history of its client's operations without help from me (that's the intuitive way), but it chooses to rely on my API, the reason being is that I asked explicitly that the data be stored on my end so it could be used in the future (after cleaning) as a training database for the AI models, in the counterpart to reduce any security concerns and suspicions: "did I really delete the record that I was asked to, or I just hid them", I asked just for URL's to the images and videos so the client could delete them from the files storage he uses whenever he likes.

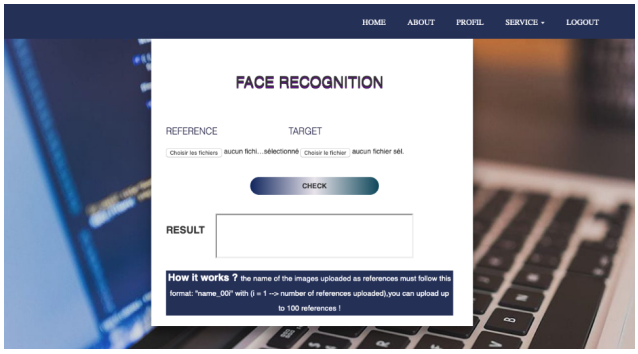
6.5 Happy customer

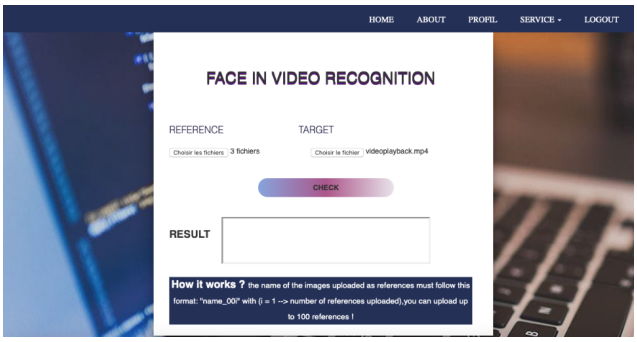





DASHBOARD
Dashboard
User
History of past operations
History of past operations
START
END
2020-09-11
2020-09-26
Search
Past operations graph
Total Claimed
Total Sum


HOME
ABOUT
PROFIL
SERVICE
LOGOUT
WRITER RECOGNITION
REFERENCE
TARGET
CHECK
RESULT
How it works ? the name of the images uploaded as references must follow this format: "name_00i" with (i = 1 --> number of references uploaded),you can upload up to 100 references !


HOME
ABOUT
PROFIL
SERVICE
LOGOUT
FACE RECOGNITION
REFERENCE
TARGET
CHECK
RESULT
How it works ? the name of the images uploaded as references must follow this format: "name_00i" with (i = 1 --> number of references uploaded),you can upload up to 100 references !


HOME
ABOUT
PROFIL
SERVICE
LOGOUT
FACE IN VIDEO RECOGNITION
REFERENCE
TARGET
3 fichiers
videoplayback.mp4
CHECK
RESULT
How it works ? the name of the images uploaded as references must follow this format: "name_00i" with (i = 1 --> number of references uploaded),you can upload up to 100 references !

Chapter 7

Recommendations for future works

In this chapter, we present some pitfalls that you may not have caught, and also some recommendations for anyone who wants to build on my work.

Just recommendations, nothing that we are 100% sure about (we didn't test it, so who we are to talk).

7.1 The Writer Identification pipeline

- If you can afford a powerful machine (GPU+CPU+RAM), doing a Grid search on all the possible intervals and options in the pipeline is highly valuable (I didn't test many combinations because of the close deadline and slow machines),
- We couldn't find the optimal number of clusters, maybe you can. We expect that using the euclidean distance there aren't any clusters (the other implementations that I checked of clustering algorithms accepting user-defined distances are extremely slow): just a big cloud of uniformly distributed points that I can't visualize considering the high dimensions of the vectors,
- We think that for SIFT and A-KAZE active preprocessing is a bad idea (it is already integrated), but inspecting doing it for the autoencoder might be a good idea,
- The bare-metal architecture of the autoencoder is written for MNIST noise removal, inspecting the state of the art architectures and implementing some prototypes might be beneficial.
- Imitation detection is a nice topic to look into.

7.2 The API

- Migrate From Flask to FastAPI. We didn't know of the latter framework until it was too late: use it, it's better in the long run,
- Document the API, we don't mean just via a readme but using something like swagger (If you use FastAPI as I recommend it comes integrated),
- Secure the API by enforcing authentication and checking authorization,
- Instead of a monolithic architecture, where you have to redeploy the entire application each time you integrate a new service or update another one, hitting a limit of computational resources that you can add because you have to scale vertically instead of horizontally (the other choice is to use a load balancer), use a micro-services architecture. We won't go through why, when, and how, you know the keyword and have a good article here [74].

We are aware that some of these points are stand-alone projects, we had to write it in case you need/want to continue with our work.

Thanks.

Bibliography

[1] European Commission. Payment Services Directive (PSD2): Regulatory Technical Standards (RTS) enabling consumers to benefit from safer and more innovative electronic payments. https://ec.europa.eu/commission/presscorner/detail/en/MEMO_17_4961, 27-Nov-2017. [Online; accessed 24-Sep-2020].

[2] Melanie Maynes. One simple action you can take to prevent 99.9 percent of attacks on your accounts. https://www.microsoft.com/security/blog/2019/08/20/one-simple-action-you-can-take-to-prevent-99-9-percent-of-account-attacks/, 20-Aug-2019. [Online; accessed 24-Sep-2020].

[3] David Smith. How Secure Is Behavioral Biometrics? https://www.infosecurity-magazine.com/opinions/secure-behavioral-biometrics/, 18-Oct-2018. [Online; accessed 24-Sep-2020].

[4] International Biometrics + Identity Association. Behavioral Biometrics white paper. https://www.ibia.org/download/datasets/3839/Behavioral%20Biometrics%20white%20paper.pdf. [Online; accessed 24-Sep-2020].

[5] Lambert Schomaker. Writer identification and verification. *Advances in Biometrics: Sensors, Algorithms and Systems*, 2008.

[6] Jane Lewis. *Forensic Document Examination: Fundamentals and Current Trends*. Academic Press; 1st Edition, 2014.

[7] Constantin Papaodysseus, P. Rousopoulos, F. Giannopoulos, Solomon Zannos, Dimitris Arabadjis, Michail Panagopoulos, E. Kalfa, Christopher Blackwell, and S. Tracy. Identifying the writer of ancient inscriptions and byzantine codices. a novel approach. *Computer Vision and Image Understanding*, 121, 04 2014.

[8] Andrew M. Colman. *A Dictionary of Psychology*. Oxford University Press, 2015.

[9] Barry Beyerstein. QA by the neuroscientist Barry Beyerstein investigates the use of handwriting analysis as a tool for psychological measurement. . https://web.archive.org/web/20070220080111/https://www.pbs.org/safarchive/3_ask/archive/qna/3282_bbeyerstein.html. [Online; accessed 24-Sep-2020].

[10] Clark Sellers. The handwriting evidence against hauptmann. (6):874–886, 1937.

[11] Diana Harrison, Ted M Burkes, and Danielle P Seiger. Handwriting examination: Meeting the challenges of science and the law. 11(4), 2009.

[12] Association of Forensic Document Examiners. FAQ of the AFDE. https://afde.org/AFDE-faq.html. [Online; accessed 24-Sep-2020].

[13] Andreas Schlapbach, Marcus Liwicki, and Horst Bunke. A writer identification system for on-line whiteboard data. *Pattern recognition*, 41:2381–2397, 2008.

[14] Lambert Schomaker. Advances in writer identification and verification. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1268–1273. IEEE, 2007.

[15] Vincent Christlein, David Bernecker, Andreas Maier, and Elli Angelopoulou. Offline writer identification using convolutional neural network activation features. In *German Conference on Pattern Recognition*, pages 540–552. Springer, 2015.

[16] Vladimir Pervouchine and Graham Leedham. Extraction and analysis of forensic document examiner features used for writer identification. *Pattern Recognition*, 40(3):1004–1013, 2007.

[17] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chellappa, and Matti Pietikäinen. Bow meets cnn: Two decades of texture representation.

[18] Chawki Djeddi, Somaya Al-Maadeed, Imran Siddiqi, Gattal Abdeljalil, Sheng He, and Younes Akbari. Icfhr 2018 competition on multi-script writer identification. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 506–510. IEEE, 2018.

[19] wikipedia. International Conference on Document Analysis and Recognition. https://en.wikipedia.org/wiki/International_Conference_on_Document_Analysis_and_Recognition. [Online; accessed 24-Sep-2020].

[20] 16th International Conference on Document Analysis and Recognition ICDAR 2021 webite. https://icdar2021.org/. [Online; accessed 24-Sep-2020].

[21] 17th International Conference on Frontiers in Handwriting Recognition 2020 website. http://icfhr2020.tu-dortmund.de/. [Online; accessed 24-Sep-2020].

[22] Florence Cloppet, Véronique Eglin, Dominique Stutzmann, Nicole Vincent, et al. Icfhr2016 competition on the classification of medieval handwritings in latin script. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 590–595. IEEE, 2016.

[23] Vincent Christlein, Martin Gropp, Stefan Fiel, and Andreas Maier. Unsupervised feature learning for writer identification and writer retrieval. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 991–997. IEEE, 2017.

[24] Fotini Simistira, Manuel Bouillon, Mathias Seuret, Marcel Würsch, Michele Alberti, Rolf Ingold, and Marcus Liwicki. Icdar2017 competition on layout analysis for challenging medieval manuscripts. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1361–1370. IEEE, 2017.

[25] Chris Tensmeyer, Daniel Saunders, and Tony Martinez. Convolutional neural networks for font classification. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 985–990. IEEE, 2017.

[26] Stefan Fiel, Florian Kleber, Markus Diem, Vincent Christlein, Georgios Louloudis, Stamatopoulos Nikos, and Basilis Gatos. Icdar2017 competition on historical document writer identification (historical-wi). In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1377–1382. IEEE, 2017.

[27] Gattal Abdeljalil, Chawki Djeddi, Imran Siddiqi, and Somaya Al-Maadeed. Writer identification on historical documents using oriented basic image features. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 369–373. IEEE, 2018.

[28] ICDAR. ICDAR2019 COMPETITION ON IMAGE RETRIEVAL FOR HISTORICAL HANDWRITTEN DOCUMENTS. https://clamm.irht.cnrs.fr/icdar2019-hdrc-ir/. [Online; accessed 24-Sep-2020].

[29] Songxuan Lai and Lianwen Jin. Offline writer identification based on the path signature feature. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1137–1142. IEEE, 2019.

[30] Arshia Rehman, Saeeda Naz, and Muhammad Imran Razzak. Writer identification using machine learning approaches: a comprehensive review. *Multimedia Tools and Applications*, 78(8):10889–10931, 2019.

[31] Chayan Halder, Sk Md Obaidullah, and Kaushik Roy. Offline writer identification and verificationa state-of-the-art. In *Information Systems Design and Intelligent Applications*, pages 153–163. Springer, 2016.

[32] Sameh M Awaida and Sabri A Mahmoud. State of the art in off-line writer identification of handwritten text and survey of writer identification of arabic text. *Educational Research and Reviews*, 7(20):445–463, 2012.

[33] Tony Lindeberg. Image matching using generalized scale-space interest points. *Journal of mathematical Imaging and Vision*, 52(1):3–36, 2015.

[34] University of British Columbia. patent for Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image. https://patents.google.com/patent/US6711293B1/en. [Online; accessed 24-Sep-2020].

[35] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[36] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[37] OpenCV documentation. Introduction to SIFT (Scale-Invariant Feature Transform). https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html. [Online; accessed 24-Sep-2020].

[38] Minghao Ning. SIFT(Scale-invariant feature transform). https://towardsdatascience.com/sift-scale-invariant-feature-transform-c7233dc60f37, 14-Apr-2019. [Online; accessed 24-Sep-2020].

[39] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *European Conference on Computer Vision*, pages 214–227. Springer, 2012.

[40] Pablo F Alcantarilla and T Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[41] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[42] AHMED FAWZY GAD. Image Compression Using Autoencoders in Keras. https://blog.paperspace.com/autoencoder-image-compression-keras, 31-Jan-2020. [Online; accessed 24-Sep-2020].

[43] Nianyin Zeng, Hong Zhang, Baoye Song, Weibo Liu, Yurong Li, and Abdullah M Dobaie. Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing*, 273:643–649, 2018.

[44] Eduard Pallàs Arranz. Unsupervised feature learning for writer identification. Master's thesis, Universitat Politècnica de Catalunya, 2018.

[45] Wikipedia. k-means clustering. https://en.wikipedia.org/wiki/K-means_clustering, 24-Sep-2020. [Online; accessed 24-Sep-2020].

[46] Se-Hoon Jung, Hansung Lee, and Jun-Ho Huh. A novel model on reinforce k-means using location division model and outlier of initial value for lowering data cost. *Entropy*, 22(8):902, 2020.

[47] Chris Piech. K-Means. https://stanford.edu/~cpiech/cs221/handouts/kmeans.html. [Online; accessed 24-Sep-2020].

[48] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.

[49] wikipedia. k-medoids. https://en.wikipedia.org/wiki/K-medoids, 23-Aug-2020. [Online; accessed 24-Sep-2020].

[50] Has QUIT-Anony-Mousse. What makes the distance measure in k-medoid better than k-means? https://stackoverflow.com/a/21646392, 17-Mar-2019. [Online; accessed 24-Sep-2020].

[51] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE, 2010.

[52] Manabu Okawa. From bovw to vlad with kaze features: Offline signature verification considering cognitive processes of forensic experts. *Pattern Recognition Letters*, 113:75–82, 2018.

[53] Victor Lavrenko. Principal component analysis for the impatient. https://youtu.be/QP43Iy-QQWY, 15-Sep-2015. [Online; accessed 24-Sep-2020].

[54] Wei Yang, Luhui Xu, Xiaopan Chen, Fengbin Zheng, and Yang Liu. Chi-squared distance metric learning for histogram data. *Mathematical Problems in Engineering*, 2015, 2015.

[55] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *European conference on computer vision*, pages 364–378. Springer, 2008.

[56] Hucker Marius. Tree algorithms explained: Ball Tree Algorithm vs. KD Tree vs. Brute Force. https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940, 15-Jun-2020. [Online; accessed 24-Sep-2020].

[57] Steve Hanov. VP trees: A data structure for finding stuff fast. http://stevehanov.ca/blog/?id=130, 2012. [Online; accessed 24-Sep-2020].

[58] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[60] OpenCV team. About OpenCV. https://opencv.org/about/. [Online; accessed 24-Sep-2020].

[61] Wikipedia. Keras. https://en.wikipedia.org/wiki/Keras, 21-Sep-2020. [Online; accessed 24-Sep-2020].

[62] U. Marti and H. Bunk. To download the IAM Handwriting Database. http://www.fki.inf.unibe.ch/databases/iam-handwriting-database/download-the-iam-handwriting-database. [Online; accessed 24-Sep-2020].

[63] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.

[64] AA Brink, RMJ Niels, RA Van Batenburg, CE Van den Heuvel, and LRB Schomaker. To download the TrigraphSlant Database. http://unipen.org/trigraphslant.html. [Online; accessed 24-Sep-2020].

[65] AA Brink, RMJ Niels, RA Van Batenburg, CE Van den Heuvel, and LRB Schomaker. Towards robust writer verification by correcting unnatural slant. *Pattern Recognition Letters*, 32(3):449–457, 2011.

[66] Abdelaali Hassaine. To download the ICDAR 2013 - Gender Identification Competition Dataset. http://tc11.cvc.uab.es/datasets/GenderIdentifify2013_1, 25-Jun-2015. [Online; accessed 24-Sep-2020].

[67] Abdelâali Hassaïne, Somaya Al Maadeed, Jihad Aljaam, and Ali Jaoua. Icdar 2013 competition on gender prediction from handwriting. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1417–1421. IEEE, 2013.

[68] Adrian Rosebrock. Autoencoders for Content-based Image Retrieval with Keras and TensorFlow. https://en.wikipedia.org/wiki/Keras, 30-Mar-2020. [Online; accessed 24-Sep-2020].

[69] Pratap Dangeti. *Statistics for machine learning*. Packt Publishing Ltd, 2017.

[70] Selecting the number of clusters with silhouette analysis on KMeans clustering. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html. [Online; accessed 24-Sep-2020].

[71] Flask README description of the project. https://github.com/pallets/flask, 21-Jul-2020. [Online; accessed 24-Sep-2020].

[72] MongoBD official website. https://www.mongodb.com/. [Online; accessed 24-Sep-2020].

[73] MongoBD Atlas official documentation. https://docs.atlas.mongodb.com/. [Online; accessed 24-Sep-2020].

[74] James Lewis Martin Fowler. Microservices. https://martinfowler.com/articles/microservices.html, 25-Mar-2014. [Online; accessed 24-Sep-2020].