



Report

# Reverse Image Search Based on AutoEncoders for the MNIST Dataset

*Author :*
BARKOUKI Oussama

*Supervisor :*
Mr. Chiheb Raddouane

College Year 2020 - 2021

# ACKNOWLEDGMENTS

First and foremost, we thank God for providing us with the strength and courage to continue out our task despite all hurdles and problems. Pr. Chiheb Raddouane, who monitored the development of this project step by step, deserves our appreciation. We appreciate his availability, participation, and invaluable guidance in helping us stay on track and accomplish such positive outcomes.

We hope that this text accurately portrays our effort and correctly shows the various aspects that we had to address.

Sincerely, Oussama Barkouki.

# ABSTRACT

This report covers the project that was completed as part of the semester's final project, which involved using auto-encoders to extract latent characteristics from photos and apply them for reverse image search.

The goal of this project is to create an auto-encoder that will encapsulate the latent representation of photos and utilise big data approaches to compare and sort photographs submitted by users.

Before proceeding with the implementation, it was required to first have a thorough grasp of the fundamentals of auto-encoders.

# TABLE DES FIGURES

# TABLE DES MATIÈRES

# GENERAL INTRODUCTION

As we get closer to a digital world, cybersecurity is becoming increasingly important. When it comes to digital security, the most difficult task is detecting unusual activities.

When we do any kind of online transaction, a large number of individuals prefer to use credit cards. Credit card credit limits can sometimes assist us in making purchases even if we do not have the funds available at the moment. Cyber attackers, on the other hand, take use of these qualities.

To address this issue, we need a system that can stop a transaction if it detects something suspicious.

This necessitates the development of a system that can track the pattern of all transactions and, if any pattern is aberrant, terminate the transaction.

This report summarizes the various stages of our project. It is divided into three sections, as follows : The first chapter covers the host organization's presentation, as well as the project's aims and objectives, as well as the strategy taken. The project's theoretical foundation and research technique are discussed in the second chapter. The project's realization is the subject of the third chapter. We'll wrap off with a personal evaluation of the project.

CHAPITRE

1

# GENERAL CONTEXT OF THE PROJECT

# 1 Introduction

This chapter's main goal is to introduce the project's low-level keywords, which range from auto-encoders to reverse image search.

# 2 Reverse Image Search

The sample picture is what formulates a search query in terms of information retrieval. Reverse image search is a content-based image retrieval (CBIR) query strategy that entails giving the CBIR system with a sample image on which it will then base its search. Reverse image search, in particular, is characterised by a scarcity of search phrases.

This basically eliminates the need for a user to make educated guesses about keywords or concepts that may or may not yield a correct result. Users may also utilise reverse image search to find material that is relevant to a given example image.

The so-called "reverse image search" is an essential application within content-based image retrieval. This is the application of computer vision techniques to the image retrieval issue, i.e., finding images in vast databases. The term "content-based" refers to a search that examines the image's content rather than its information, such as keywords, tags, or descriptions.

# 3 Auto-encoders

Autoencoders are a sort of neural network in which the input and output are the same. They compress the input into a lower-dimensional latent space before reconstructing the output from this representation. The latent representation of the input is a concise "summary" or "compression" of the input.

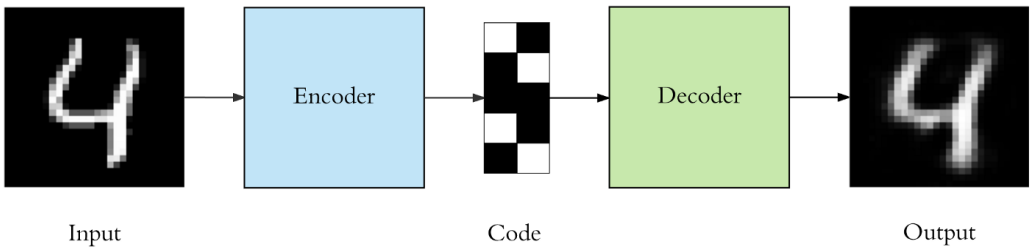


FIGURE 1.1 – Auto-encoder

Autoencoders are primarily dimensionality reduction (or compression) algorithms with a few key characteristics :

- Data-specific : Autoencoders can only significantly compress data that is comparable to what they were trained on.
- Lossy : The autoencoder's output will not be precisely the same as the input ; instead, it will be a near but deteriorated representation.
- Autoencoders are classified as unsupervised learning techniques since they do not require explicit labels to train on.

Both the encoder and decoder are feedforward neural networks that are completely coupled. The code is a single layer of an ANN with the dimensions of our choice. We specify the number of nodes in the code layer (code size) before training the autoencoder.

To generate the code, the input is first sent through the encoder. The output is subsequently produced only by the decoder using the code. The objective is to produce an output that is identical to the input. It should be noted that the decoder design is the inverse of the encoder architecture. This is not a must, although it is often the case. The only criterion is that the input and output dimensions be the same. Anything in the centre may be manipulated.

CHAPITRE

# 2

# EXISTING SOLUTIONS

# Introduction

This chapter will provide a survey of the literature, allowing us to describe and define all of the previous work on this field

The majority of search engines available today are textual, keyword-based, or metadata-based. To find the related image, one or more keywords must be specified. The inherent disadvantages of text-based, keyword-based, or metadata-based picture search are as follows :

— Manually annotating keywords for web images is a time-consuming process.

— Annotation may be inconsistent owing to numerous contents in a single image

— it is also difficult for different indexers to provide precisely the same annotations

In the 1990s, content-based image retrieval was proposed as a solution to the challenges faced with keyword-based picture search.

A previous strategy used text-based meta-search to generate an initial digital picture collection with a high recall rate but a low accuracy rate. The picture content-based processing is then used to generate a significantly more relevant result.

Another system re-ranks the results of a normal web-based picture search engine in order of relevancy. The system is unique in that it just takes a keyword query from the user and infers a feature space representation of that query, which is then used to re-rank the results.

The CBIR analysis result is Reverse Image Search. The Reverse Picture Search feature allows the user to identify the information connected with a selected sample image, the quality of an image, uncover altered copies and derivative works, and also a precise match of the input image.

CHAPITRE

# 3

# PROPOSED SOLUTION

# Introduction

This chapter will present the proposed solution.

## 1 Dataset

Before we go into the suggested approach, we need to discuss about the dataset that we utilised to create this project. We first planned to utilise the imagenet dataset, however owing to hardware limitations, we switched to the mnist dataset.

The Modified National Institute of Standards and Technology dataset is abbreviated as the MNIST dataset. It is a collection of 60,000 little square 2828 pixel grayscale pictures of handwritten single numbers ranging from 0 to 9.

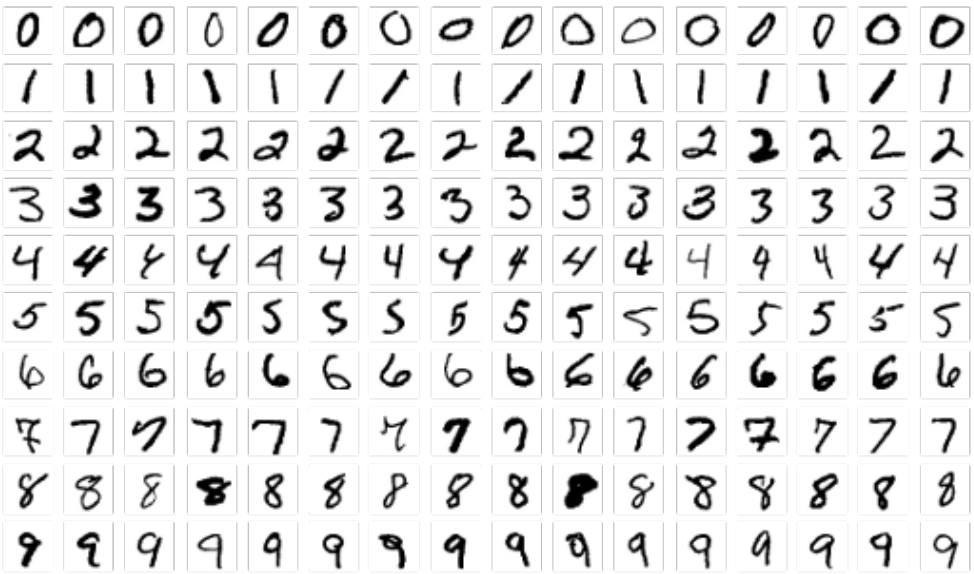

FIGURE 3.1 – Mnist Dataset

## 2 Proposed Auto-encoder Architecture

While looking for an appropriate architecture to capture the latent properties of our dateset, we discovered that combining Convolutional neural networks with our auto-encoder will be really beneficial.

Convolutional neural networks are capable of collecting and recognising essential elements in our data, which is why they are best suited to our objective of extracting information/context from photos.

Convolutions are then employed successively while down sampling until a vector with 128 dimensions is obtained. That's it for the encoder. We used convolutions again for the decoder, but this time with an upsampling layer between each one to increase the size of our data and rebuild the original image.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2D  (None, 14, 14, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 8)         1160

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 8)          0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 8)           584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 8)          0
 2D)

 flatten (Flatten)           (None, 128)               0

=================================================================
Total params: 1,904
Trainable params: 1,904
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 3.2 – Encoder

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 4, 4, 8)           0

 conv2d_3 (Conv2D)           (None, 4, 4, 8)           584

 up_sampling2d (UpSampling2D  (None, 8, 8, 8)          0
 )

 conv2d_4 (Conv2D)           (None, 8, 8, 8)           584

 up_sampling2d_1 (UpSampling  (None, 16, 16, 8)        0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 16)        1168

 up_sampling2d_2 (UpSampling  (None, 28, 28, 16)       0
 2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 1)         145

=================================================================
Total params: 2,481
Trainable params: 2,481
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 3.3 – Decoder

## 2.1 Training

We utilized the Adam optimizer with a binary cross-entropy loss to train the model. The model is trained by feeding it input and expecting the output to be identical to the input.

While fine-tuning the model's hyper-parameters, we discovered that after 100 epochs, the model stops learning and accuracy no longer improves.

The graphs below depict the progression of accuracy as well as loss over time.

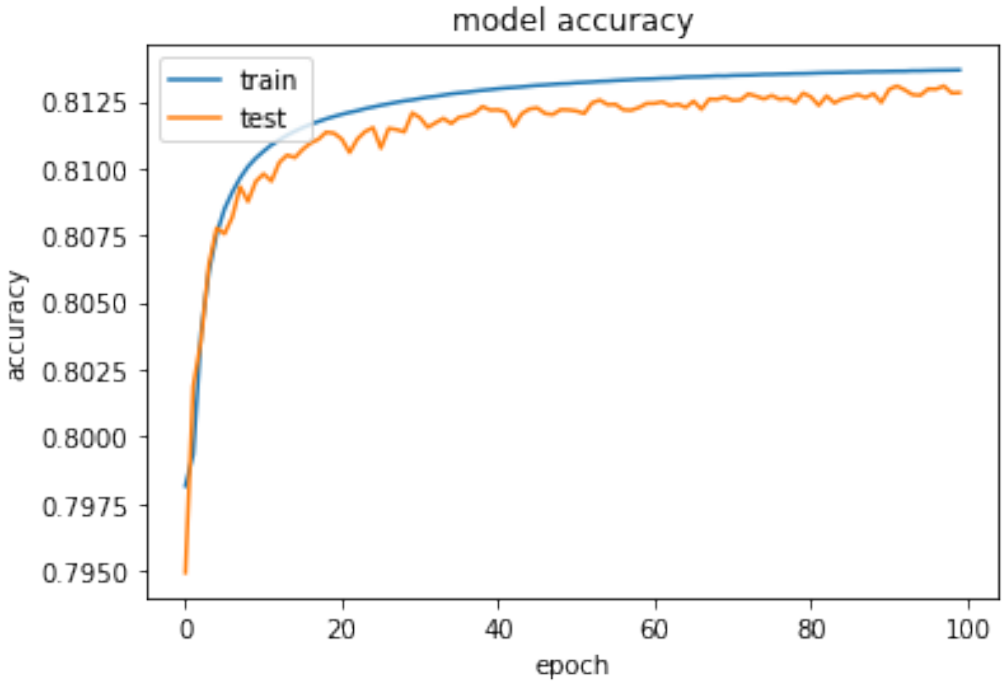


FIGURE 3.4 – Accuracy

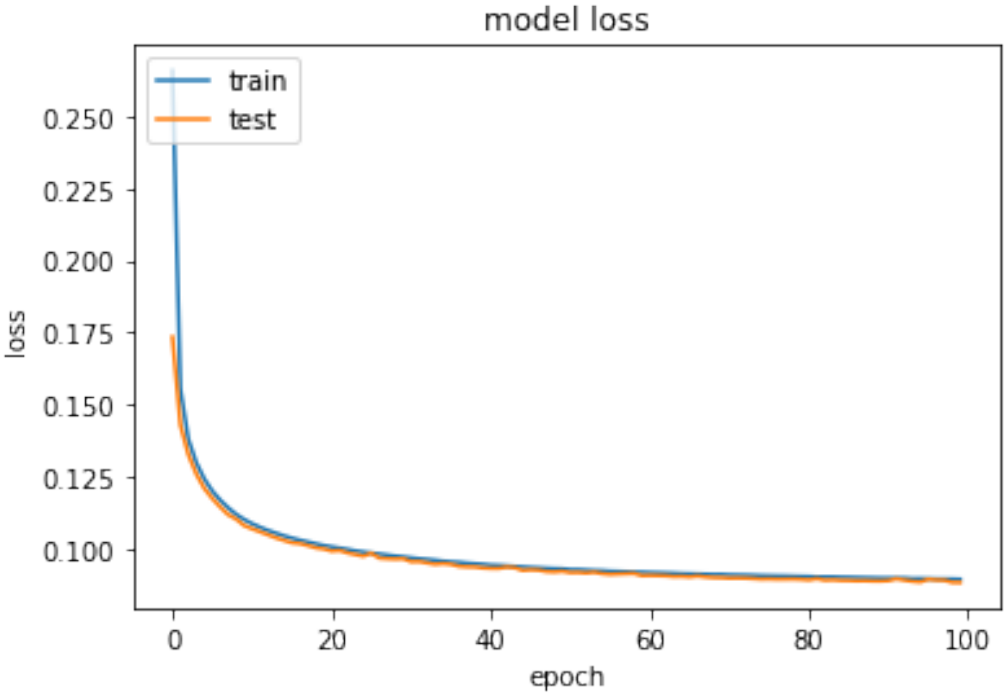


FIGURE 3.5 – Loss

# 3 Image Search

We split the auto-encoder after training it and use only the encoder component to produce embedding for all of the pictures in our dataset. The embedding allows us to simply apply a euclidean distance function to compare photos. In this example, we utilized Kdtree, which is an algorithm that can compute the distance between a data sample and all of our datasets and rank them based on the distance while picking the topN.

For example we took the image bellow :

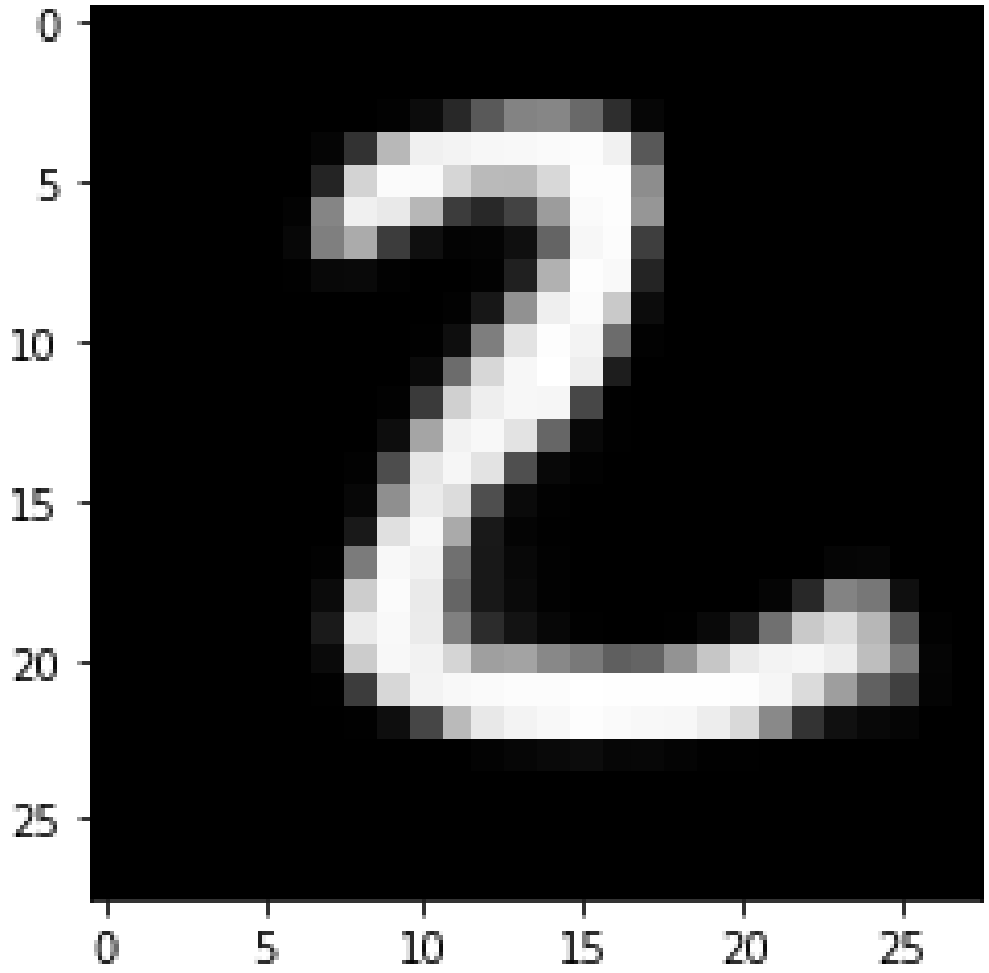


FIGURE 3.6 – Data Sample

Following the application of everything we've covered thus far and the selection of the top ten neighbors, our approach resulted in the following.

FIGURE 3.7 – 10 neighbors

# 4 Web Application

## 4.1 tools

### 4.1.1 Cassandra

Apache Cassandra is an open source NoSQL distributed database trusted by thousands of companies for scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data.


FIGURE 3.8 – Cassandra

### 4.1.2 Docker

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine.

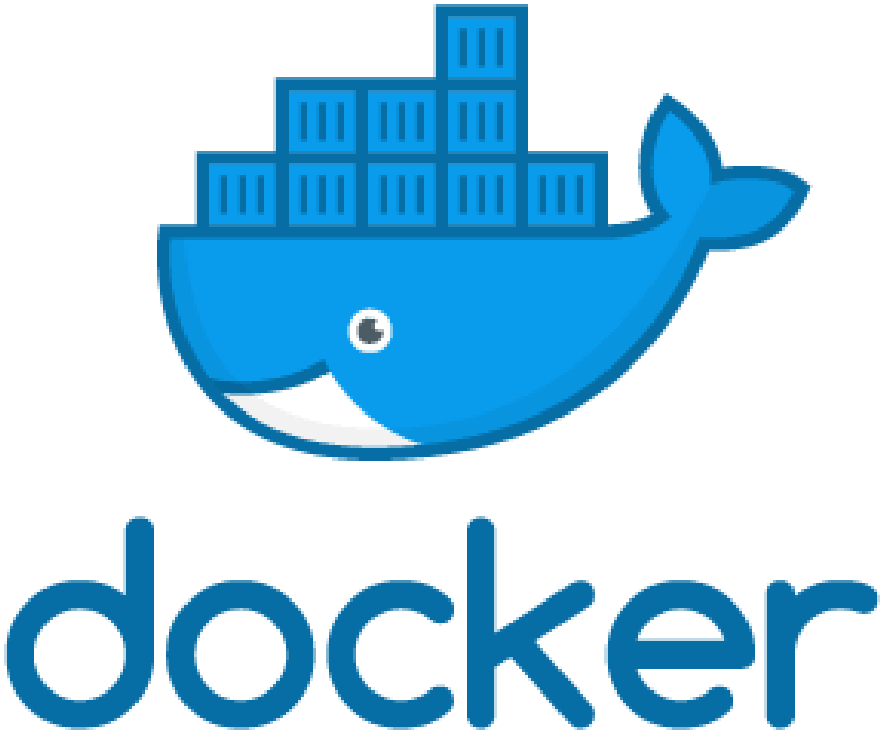


FIGURE 3.9 – Docker

### 4.1.3 FastAPI

FastAPI is a Web framework for developing RESTful APIs in Python. FastAPI is based on Pydantic and type hints to validate, serialize, and deserialize data, and automatically auto-generate OpenAPI documents. It fully supports asynchronous programming and can run with Uvicorn and Gunicorn.


FIGURE 3.10 – FastAPI

### 4.1.4 React js

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

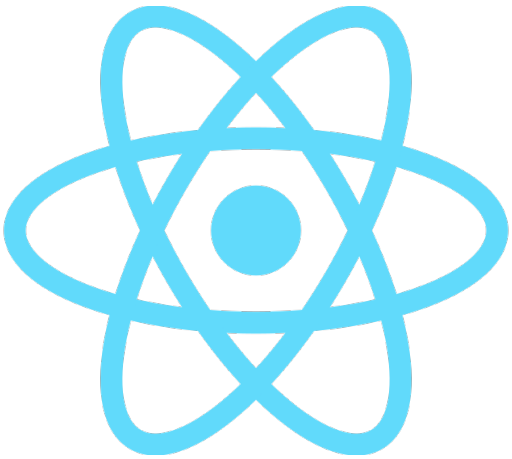

FIGURE 3.11 – FastAPI

## 4.2

Web Interfaces

## 4.3 Seach Input




FIGURE 3.12 – Input

## 4.4 Search Results

Given the following image :

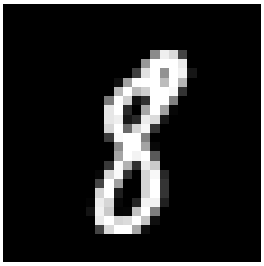

FIGURE 3.13 – Test Sample

The results were these :

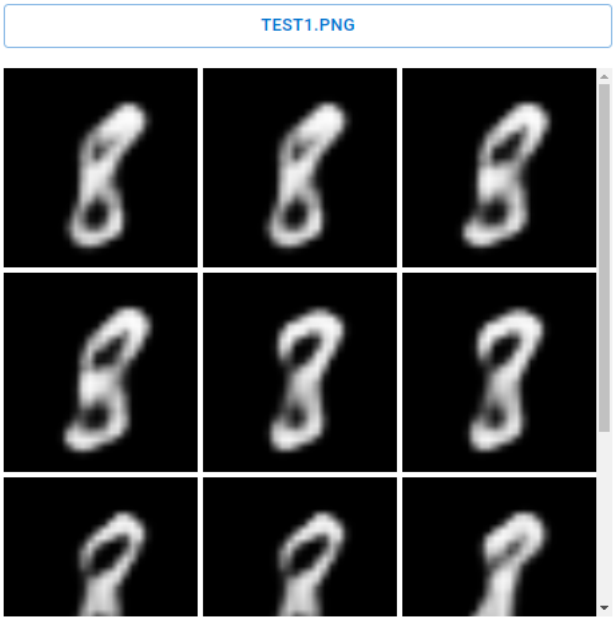


FIGURE 3.14 – Results

# 5 Final Results

In conclusion, the project was rather simple, with the main difficult aspect being expanding the proposed solution to operate with bigger datasets and serve several customers at the same time. As a result, big data approaches and MLOps were used to deliver the solution. More optimization is necessary if the model is to be successfully deployed.

Overall, it was a fantastic learning opportunity to combine Machine Learning with computer engineering.

# BIBLIOGRAPHIE

[1] https ://github.com/serengil/tensorflow-101/blob/master/python/Cassandra-Face-Recognition.ipynb

[2] http ://yann.lecun.com/exdb/mnist/

[3] https ://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html