# Rapport de projet de Vision par Ordinateur

## Tri postal - Reconnaissance OCR de codes postaux par KNN

**Jean-Eudes CODO**

3e année IA

**Date :** Février 2026

# Table des matières

# 1 Présentation du Projet

## 1.1 Contexte

Le tri postal automatique nécessite la reconnaissance optique de caractères (OCR) pour lire les codes postaux inscrits sur les enveloppes. Ce projet implémente un système complet de reconnaissance de chiffres, depuis l'acquisition de l'image jusqu'à la classification, en utilisant l'algorithme des **k plus proches voisins (k-NN)** codé entièrement from scratch, sans bibliothèque de machine learning.

## 1.2 Objectif

L'objectif est de construire un pipeline capable de :

1. **Prétraiter** une image de code postal (binarisation)
2. **Segmenter** les chiffres individuels (détection de contours)
3. **Extraire** un vecteur de caractéristiques discriminant pour chaque chiffre
4. **Classifier** chaque chiffre parmi les classes 0–9
5. **Évaluer** les performances via des métriques standards

## 1.3 Données

— **Entraînement :** 10 images (`0.png` à `9.png`), chacune contenant 5 exemplaires du chiffre correspondant ⇒ **50 échantillons** au total

— **Test :** 10 images de codes postaux — 5 images du code `59130` et 5 images du code `62487` ⇒ **50 chiffres** à classifier

## 1.4 Code source

Le projet est également disponible sur GitHub via le lien suivant :
https://github.com/Eudoo/Postal_sorting.git.

# 2 Architecture du Système

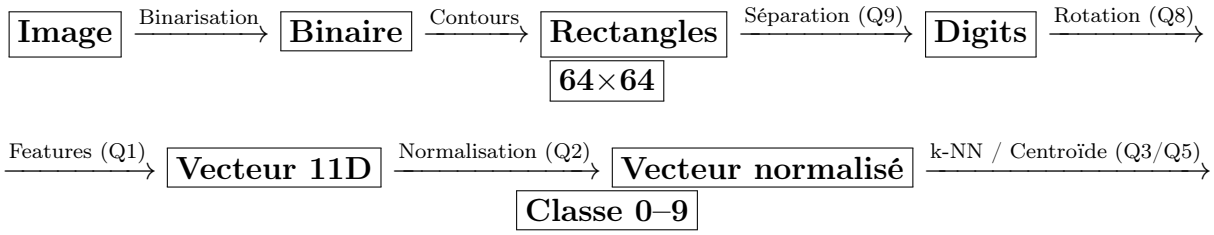
## 2.1 Organisation des fichiers

Le projet est organisé en deux phases distinctes :

| Dossier | Fichier | Rôle |
|---|---|---|
| `offline/` | `utils.py` | Fonctions utilitaires (prétraitement, features, normalisation) |
| | `labeling.py` | Script d'entraînement |
| | `knn_data.npz` | Modèle sauvegardé |
| `online/` | `inference.py` | Script de prédiction (k-NN / centroïde / les deux) |
| | `knn_utils.py` | Algorithmes de classification |
| | `metrics.py` | Fonctions d'évaluation |
| `images/` | `digits/` | Images d'entraînement (0.png à 9.png) |
| | `postal_code/` | Images de test |
| | `results/` | Images annotées des prédictions |

## 2.2 Pipeline de traitement

Le pipeline complet suit le schéma suivant :




# 3 Solution Proposée

## 3.1 Espace de décision (Q1, Q7)

### 3.1.1 Prétraitement

Chaque image est convertie en niveaux de gris puis binarisée par seuillage d'Otsu (`THRESH_BINARY_INV`), ce qui produit une image où les pixels du chiffre sont blancs (255) et le fond est noir (0).

### 3.1.2 Segmentation

Les contours externes sont détectés via `cv2.findContours` en mode `RETR_EXTERNAL`. Les rectangles englobants sont filtrés (aire $> 20$ pixels pour éliminer le bruit) puis triés par lignes (coordonnée $y$) et par position horizontale ($x$). Chaque chiffre est extrait, redimensionné à $64 \times 64$ pixels et re-binarisé.

### 3.1.3 Descripteurs de cavités — Masques de visibilité murale (Q1)

Le cœur de l'espace de décision repose sur l'analyse des **cavités** du chiffre. Pour chaque pixel de fond, on détermine s'il est « bloqué » par un pixel du chiffre dans chaque direction cardinale (Nord, Sud, Est, Ouest) :

— **Mur Nord** (`wall_north`) : vrai si un pixel blanc existe *au-dessus* dans la même colonne

— **Mur Sud** (`wall_south`) : vrai si un pixel blanc existe *en-dessous*

— **Mur Est** (`wall_east`) : vrai si un pixel blanc existe *à droite* sur la même ligne

— **Mur Ouest** (`wall_west`) : vrai si un pixel blanc existe *à gauche*

En combinant ces 4 masques par opérations logiques, on définit 5 zones de cavités :

| **Cavité** | **Définition** | **Exemple** |
|---|---|---|
| Centrale | Bloqué N + S + E + W (trou fermé) | Intérieur du 0, du 8 |
| Nord | Ouvert au Nord, bloqué S + E + W | — |
| Sud | Ouvert au Sud, bloqué N + E + W | — |
| Est | Ouvert à l'Est, bloqué N + S + W | Ouverture haute du 6 |
| Ouest | Ouvert à l'Ouest, bloqué N + S + E | Ouverture basse du 5 |

Pour chaque cavité, on extrait :

— La **surface** (normalisée par l'aire de l'image)

— Le **barycentre Y** (normalisé par la hauteur de l'image) — pour les cavités directionnelles

— Le **nombre de blocs connectés** / 2 — pour la cavité centrale (via `cv2.connectedComponents`)

#### 3.1.4 Descripteur de solidité (Q7)

En complément des cavités, un descripteur *regionprops* invariant à l'échelle est ajouté :

$$\text{Solidité} = \frac{\text{Aire du contour}}{\text{Aire de l'enveloppe convexe}} \tag{1}$$

Ce ratio, compris entre 0 et 1, mesure la « compacité » de la forme. Un chiffre comme le **0** aura une solidité élevée ($\approx 1$), tandis qu'un **1** sera plus faible.

#### 3.1.5 Vecteur de caractéristiques final

Le vecteur final comporte **11 features** :

| # | **Feature** | **Plage** |
|---|---|---|
| 1 | Surface cavité centrale | $[0, 1]$ |
| 2 | Nb blocs connectés centraux / 2 | $\{0,\ 0.5,\ 1.0\}$ |
| 3 | Surface cavité Nord | $[0, 1]$ |
| 4 | Barycentre Y cavité Nord | $[0, 1]$ |
| 5 | Surface cavité Sud | $[0, 1]$ |
| 6 | Barycentre Y cavité Sud | $[0, 1]$ |
| 7 | Surface cavité Est | $[0, 1]$ |
| 8 | Barycentre Y cavité Est | $[0, 1]$ |
| 9 | Surface cavité Ouest | $[0, 1]$ |
| 10 | Barycentre Y cavité Ouest | $[0, 1]$ |
| 11 | Solidité (Q7) | $[0, 1]$ |

## 3.2 Apprentissage hors-ligne (Q2)

L'entraînement consiste à :

1. Parcourir les 10 images d'entraînement (`0.png` à `9.png`)
2. Pour chaque image : binariser → détecter les contours → séparer les chiffres collés (Q9) → extraire les features
3. Construire la matrice $X \in \mathbb{R}^{50\times 11}$ et le vecteur $y \in \{0,\ldots,9\}^{50}$
4. Appliquer une **normalisation Min-Max** :

$$X_{\text{norm}}^{(i,j)} = \frac{X^{(i,j)} - \min_j}{\max_j - \min_j} \tag{2}$$

Cette normalisation ramène chaque feature dans $[0,1]$, évitant qu'une dimension à grande échelle domine le calcul de distance euclidienne.

Enfin, les **centroïdes** (vecteur moyen par classe) sont calculés sur les données normalisées :

$$\boldsymbol{c}_k = \frac{1}{|C_k|}\sum_{i\in C_k} \boldsymbol{x}_i \qquad \text{pour } k = 0,\ldots,9 \tag{3}$$

Le tout est sauvegardé dans `knn_data.npz` : $X_{\text{norm}}$, $y$, min, max, centroïdes.

## 3.3 Classification par k-NN (Q3)

Le classifieur k-NN est implémenté entièrement from scratch :

1. Calcul de la **distance euclidienne** entre le vecteur test et chaque échantillon d'entraînement :

$$d(\boldsymbol{x}, \boldsymbol{x}_i) = \sqrt{\sum_{j=1}^{11}(x_j - x_{i,j})^2} \tag{4}$$

2. Tri des distances par ordre croissant
3. Sélection des $k = 3$ plus proches voisins
4. **Vote majoritaire** : la classe la plus fréquente parmi les $k$ voisins est retournée

## 3.4 Validation (Q4)

L'évaluation utilise les métriques suivantes :

— **Précision par chiffre** : $\frac{\text{chiffres correctement classifiés}}{\text{total chiffres}}$

— **Précision par code postal** : $\frac{\text{codes postaux entièrement corrects}}{\text{total codes}}$

— **Matrice de confusion** $10 \times 10$

— **Precision par classe** : $\frac{TP}{TP+FP}$

— **Recall par classe** : $\frac{TP}{TP+FN}$

### 3.5 Classification par plus proche centroïde (Q5)

En alternative au k-NN, un classifieur par **plus proche moyenne** (nearest centroid) est implémenté :

$$\hat{y} = \arg\min_{k \in \{0,\ldots,9\}} d(\boldsymbol{x}_{\text{test}},\ \boldsymbol{c}_k) \tag{5}$$

où $\boldsymbol{c}_k$ est le centroïde de la classe $k$. Cette méthode est plus rapide (10 distances à calculer au lieu de 50) et plus stable face au bruit.

### 3.6 Comparaison k-NN vs Centroïde (Q6)

Les deux méthodes sont comparables via l'option `-mode both` du script d'inférence, qui exécute les deux classifieurs en parallèle sur les mêmes données et affiche les métriques séparément.

### 3.7 Recalage en rotation (Q8)

Les chiffres inclinés peuvent être mal classifiés (ex : un 3 penché ressemble à un 2). Pour corriger cela, on utilise les **moments centraux d'ordre 2** pour calculer l'angle de l'axe principal :

$$\theta = \frac{1}{2} \arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right) \tag{6}$$

Si l'inclinaison est comprise entre 5° et 45°, une contre-rotation est appliquée via `cv2.getRotationMatrix2D` et `cv2.warpAffine`. Cette correction est intégrée directement dans la fonction d'extraction des caractères, s'appliquant automatiquement à l'entraînement comme à l'inférence.

### 3.8 Séparation de chiffres collés (Q9)

Lorsque deux chiffres se touchent dans l'image, ils sont détectés comme un seul contour. La séparation s'effectue en 4 étapes :

1. **Détection** : si le ratio $w/h$ d'un rectangle est $\geq 1.5$, il contient probablement plusieurs chiffres
2. **Estimation** : le nombre de chiffres est estimé par round$(w/h)$
3. **Profil de projection vertical** : on calcule la somme des pixels blancs par colonne
4. **Séparation** : les points de coupure sont placés aux **vallées** (minima de la projection), qui correspondent aux zones de séparation entre chiffres

## 4 Résultats

### 4.1 Performances finales

| Méthode | Précision Chiffres | Précision Codes Postaux |
|---|---|---|
| k-NN ($k = 3$) | $47/50 =$ **94%** | $7/10 =$ **70%** |
| Centroïde | $48/50 =$ **96%** | $8/10 =$ **80%** |

### 4.2 Erreurs résiduelles

| Confusion | Méthode | Explication |
|---|---|---|
| $5 \to 7$ | k-NN + Centroïde | Chiffres allongés verticalement, cavités proches |
| $7 \to 1$ | k-NN + Centroïde | Morphologie très similaire (traits verticaux) |
| $1 \to 7$ | k-NN seulement | Introduit par la rotation ; corrigé par le centroïde |

### 4.3 Historique des versions

| Version | Feat. | k-NN Chiffres | k-NN Codes | Centr. Chiffres | Centr. Codes |
|---|---|---|---|---|---|
| Cavités seules | 10 | 94% | 70% | 94% | 70% |
| + Solidité (Q7) | 11 | 94% | 70% | **96%** | **80%** |
| + Excentricité + ratio | 13 | 92% ↓ | 70% | 94% ↓ | 70% ↓ |
| + Rotation Q8 + Sep. Q9 | 11 | 94% | 70% | **96%** | **80%** |

La version à 13 features (excentricité + ratio d'aspect) a été abandonnée car ces descripteurs ajoutaient du bruit au lieu d'information discriminante. La rotation (Q8) a permis de corriger la confusion $3 \to 2$ mais a introduit de légères régressions pour le k-NN, compensées par le centroïde.

## 5 Problèmes Rencontrés et Limites

### 5.1 Problèmes résolus

| Problème | Cause | Solution |
|---|---|---|
| Profils de bordure inefficaces | Distance au premier pixel ne mesure pas les vraies cavités | Masques de visibilité murale (N/S/E/W) |
| Scaling des features | `nb_cavities` (0–2) vs ratios (0–0.3) : échelles incompatibles | Normalisation Min-Max |
| Hybride RETR_CCOMP + masques | Méthodes hétérogènes ⇒ features incohérentes | 100% masques (approche homogène) |
| Confusion $5 \to 6$ | Cavités similaires | Ajout solidité (Q7) |
| Confusion $3 \to 2$ | Chiffre 3 incliné | Recalage rotation (Q8) |
| Excentricité dégrade | Redondance + bruit ($0 \leftrightarrow 8$) | Retirée du vecteur final |

### 5.2 Limites non résolues

— **Confusion** $5 \leftrightarrow 7$ **et** $7 \leftrightarrow 1$ : chiffres morphologiquement très proches avec nos descripteurs actuels

— **Petite base d'entraînement** : 50 échantillons (5 par classe) — peu de variabilité. Piste : data augmentation

— **Chiffres qui se touchent** : le pipeline Q9 est en place mais non activé sur les données actuelles (pas de chiffres réellement collés)

— $k$ **fixe ($k = 3$)** : choisi empiriquement, non optimisé par cross-validation

# 6 Conclusion

Ce projet a permis d'implémenter un système complet de reconnaissance de codes postaux par k-NN, from scratch. Le classifieur par centroïde atteint **96% de précision par chiffre** et **80% par code postal**, grâce à un espace de décision de 11 features basé sur les cavités (masques de visibilité murale) et la solidité.

Les principales contributions sont :

— Un espace de features **interprétable et sémantiquement riche** (cavités directionnelles, solidité)

— Deux classifieurs complémentaires : k-NN et plus proche centroïde

— Un pipeline robuste incluant la correction de rotation (Q8) et la séparation de chiffres collés (Q9)