Rapport de projet d’image

Initiation à la recherche

Ayman KACHMAR - Gabin CHARASSON - Hugo MARTIN - S4C

Table des matières

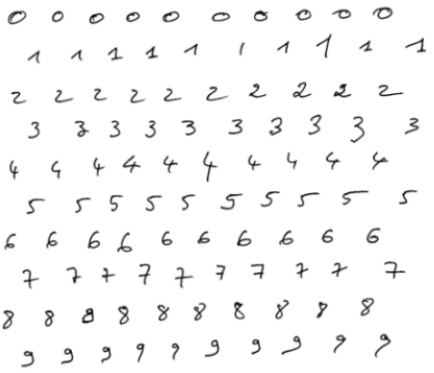
1. Introduction

a) Présentation du projet

Ce projet a pour objectif de développer des algorithmes de reconnaissance de chiffres et ainsi obtenir une évaluation de cette reconnaissance tout en utilisant une base de données fournie. Nous avons mis en place de projet en utilisant ***Python*** et des librairies pour l'utilisation d'images et de calculs mathématiques. Notre programme se nomme OCR.py.

b) Présentation de la base

La base initiale contient des images de chiffres manuscrits de 0 à 9, ayant des tailles différentes. Il y a 10 images par chiffre.



c) Enjeux scientifiques et difficultés à identifier

Ce projet concerne en partie au sujet de l'intelligence artificielle, dans lequel les humains ont besoin d'algorithmes de reconnaissances pour identifier les éléments d'une image. Ici, c'est le cas et l'avantage par rapport aux humains est que les algorithmes peuvent distinguer à l'aide de calculs mathématiques rapides les chiffres manuscrits en se basant sur des pixels.

2. Présentation d'une solution

d) Chaîne de traitement et exemples

Pour expliquer les traitements utilisés, expliquons d'abord quels choix d'algorithmes avons fait, nous avons utilisé :

- **le zoning**
- **le profil horizontal**
- **le KNN**

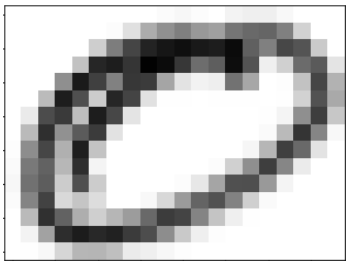
Voici comment notre équipe a décidé de procéder pour cette chaîne de traitement :

- **redimensionner** les images en 20x20 pixels afin que la définition de chaque image soit similaire, et ainsi entamer la comparaison à l'aide de matrices. Seul l'algorithme KNN a besoin d'une taille de 35x6 pixels, nous vous expliquerons pourquoi à la suite.

- **transformer** l'image en noir et blanc en appliquant un seuil de 150, ce dernier a été choisi par tâtonnement avec l'obtention de meilleurs résultats.
- **binariser** chaque image pour récupérer une matrice composée de 0 (noir) et de 1 (blanc)
- **convoluer** en normalisant les images pour modifier la plage des valeurs d'intensité des pixels noirs et blancs. Cela permet aux valeurs de la matrice binarisée d'être plus précises. Le filtre utilisé est une matrice 3x3 composée de 1/9.

Voici un extrait du code utilisant cette chaîne de traitement avant pour le fichier 0_1.png (chiffre 1) par exemple :

```
image = io.imread(name)
imMatrix= imgToBinaryMatrix(image, 35, 6)
meanKernel = np.full((3, 3), 1.0/9)
imMatrix = ndi.correlate(imMatrix, meanKernel)
imageData[name]=np.concatenate(imMatrix)
```

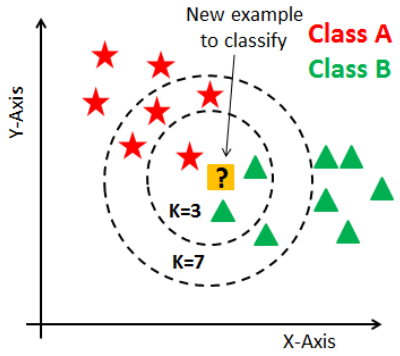


```
[[1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1. ]
[1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.9 0.8 0.7 0.7 0.8 0.9 1.  1.  1.  1. ]
[1.  1.  1.  1.  1.  1.  1.  0.9 0.8 0.7 0.6 0.4 0.3 0.3 0.4 0.6 0.8 0.9 1.  1. ]
[1.  1.  1.  1.  1.  0.9 0.8 0.6 0.4 0.3 0.2 0.1 0.  0.  0.1 0.2 0.4 0.7 0.9 1. ]
[1.  1.  1.  1.  0.9 0.7 0.4 0.2 0.1 0.  0.  0.  0.  0.  0.  0.  0.1 0.4 0.8 1. ]
[1.  1.  1.  0.9 0.7 0.3 0.1 0.  0.1 0.2 0.3 0.3 0.3 0.3 0.3 0.3 0.2 0.3 0.6 0.9]
[1.  1.  1.  0.8 0.4 0.1 0.  0.1 0.3 0.6 0.7 0.7 0.7 0.7 0.7 0.7 0.4 0.3 0.3 0.6]
[1.  1.  0.9 0.6 0.2 0.  0.1 0.3 0.7 0.9 1.  1.  1.  1.  1.  1.  0.7 0.3 0.1 0.2]
[1.  1.  0.8 0.4 0.1 0.1 0.3 0.7 0.9 1.  1.  1.  1.  1.  1.  1.  0.7 0.3 0.  0. ]
[1.  0.9 0.6 0.2 0.  0.2 0.6 0.9 1.  1.  1.  1.  1.  1.  1.  1.  0.7 0.3 0.  0. ]
[1.  0.8 0.4 0.1 0.1 0.4 0.8 1.  1.  1.  1.  1.  1.  1.  1.  0.9 0.6 0.2 0.1 0.2]
[1.  0.7 0.3 0.  0.2 0.6 0.9 1.  1.  1.  1.  1.  1.  1.  0.9 0.7 0.3 0.2 0.3 0.6]
[1.  0.7 0.3 0.  0.3 0.7 1.  1.  1.  1.  1.  1.  1.  0.9 0.7 0.3 0.1 0.2 0.6 0.9]
[1.  0.7 0.3 0.  0.3 0.7 1.  1.  1.  1.  1.  1.  0.9 0.7 0.3 0.1 0.1 0.4 0.8 1. ]
[1.  0.8 0.4 0.1 0.2 0.6 0.9 1.  1.  1.  0.9 0.8 0.6 0.3 0.1 0.1 0.3 0.7 0.9 1. ]
[1.  0.9 0.6 0.2 0.1 0.3 0.6 0.7 0.7 0.7 0.6 0.4 0.2 0.2 0.2 0.4 0.7 0.9 1.  1. ]
[1.  1.  0.8 0.4 0.1 0.1 0.2 0.3 0.3 0.3 0.2 0.2 0.2 0.4 0.6 0.8 0.9 1.  1.  1. ]
[1.  1.  0.9 0.7 0.4 0.2 0.1 0.  0.1 0.2 0.3 0.4 0.6 0.8 0.9 1.  1.  1.  1.  1. ]
[1.  1.  1.  0.9 0.8 0.6 0.4 0.3 0.4 0.6 0.7 0.8 0.9 1.  1.  1.  1.  1.  1.  1. ]
[1.  1.  1.  1.  1.  0.9 0.8 0.7 0.8 0.9 1.  1.  1.  1.  1.  1.  1.  1.  1.  1. ]]
```

La normalisation grâce au filtre montre l'intensité du niveau de gris suivant la position du pixel dans l'image et dans la matrice ci-dessus (ici les valeurs sont arrondies seulement pour cet affichage). Nous obtenons de meilleurs résultats avec cette méthode.

e) Détail de l'algorithme KNN

L'algorithme KNN (K Nearest Neighbours), consiste à faire une comparaison des voisins à une distance k de chaque pixels d'une image par rapport à un échantillon d'apprentissage, dans notre cas la totalité de la base à l'exception de l'image comparée, puis de sélectionner le candidat ayant le plus de ressemblances.



La comparaison est effectuée de la même façon que dans l'algorithme du zoning, c'est-à-dire en calculant la **distance euclidienne** de deux vecteurs.

Lors de nos expérimentations, nous avons fait tourner cet algorithme en testant différentes définitions d'images avec différentes **distances k**, nous avons pu observer qu'une image dimensionnée en 35x6 et analysée avec un k de valeur 5, on obtient 94% de réussite en parcourant l'algorithme, valeur maximale atteinte après avoir essayé chaque dimension possible avec un x variant de 0 à 100, un y variant de 0 à 26 (le programme au bout de 33h a été malencontreusement éteint), et un k variant de 1 à 31 par pas de 2 (pour utiliser des k impairs).

3. Analyse des résultats

f) Présentation de la matrice de confusion finale

```
[[10.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 10.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 10.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 10.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 10.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 10.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 10.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 10.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]
{'knn': 50, 'zn': 50, 'ph': 0}
Succès : 100.0 %
```

```
for nb in range(0,10):
            for nth in range(1,11):
                name = f'baseProjetOCR/{nb}_{nth}.png'
                alg = self.bestAlgoForNb(nb)
                self.stats[alg] += 1
                foundNb = self.guesser[alg](name)
                self.finalConfMatrix[nb][foundNb] += 1
```

Le programme choisit l'algorithme le plus à même de trouver quel chiffre renferme une image et l'applique afin d'obtenir un résultat potentiel, puis compare le nom du fichier au chiffre deviné pour construire cette matrice.

g) Analyse de cas limites

Étant donné que les résultats ne sont pas fusionnés mais comparés et sélectionnés, si aucun des 3 algorithmes n'est capable de se prononcer avec une confiance suffisante, il est quasiment sûr que la prédiction sera mauvaise. l'OCR n'est pas fiable sans échantillon suffisamment grand à l'apprentissage.

4. Conclusion

h) Résumé des résultats obtenus

Pour résumer, notre chaîne de traitement fonctionne efficacement, en tout cas, dans notre cas précis, les algorithmes les plus efficaces sont ceux de "zoning" et le "knn" qui se partagent entre deux l'ensemble des conclusions de notre OCR.

i) Perspective d'amélioration

Après réflexions, nous constatons que notre chaîne de traitement pourrait être plus élaborée pour gérer les imperfections d'une image. Par exemple, sur cette image, nous pourrions effectuer une fermeture sur l'image avec la dilatation puis l'érosion. Les trous pourraient alors disparaître et donc le trait que l'on observe au milieu du 0 ne serait pas contraignant.