

TIPE : Reconnaissance de plaques d'immatriculation

Thème : La ville

LANGLAIS Gaspard

Introduction au sujet

Problématique : Comment peut-on obtenir un système de lecture automatique des plaques d'immatriculation, fiable et rapide, applicable à un péage ?



Source : toutsurmesfinances.com

- Plusieurs centaines de péages
- Plus de 2000 parkings

Plan

1. Traitement de l'image

2. Extraction de la plaque

3. Identification des caractères

1. Traitement de l'image

Objectif : Mise en évidence des contours de l'image en utilisant un filtre Canny



Taille : 1500*846 pixels

1. Traitement de l'image : Passage en noir et blanc

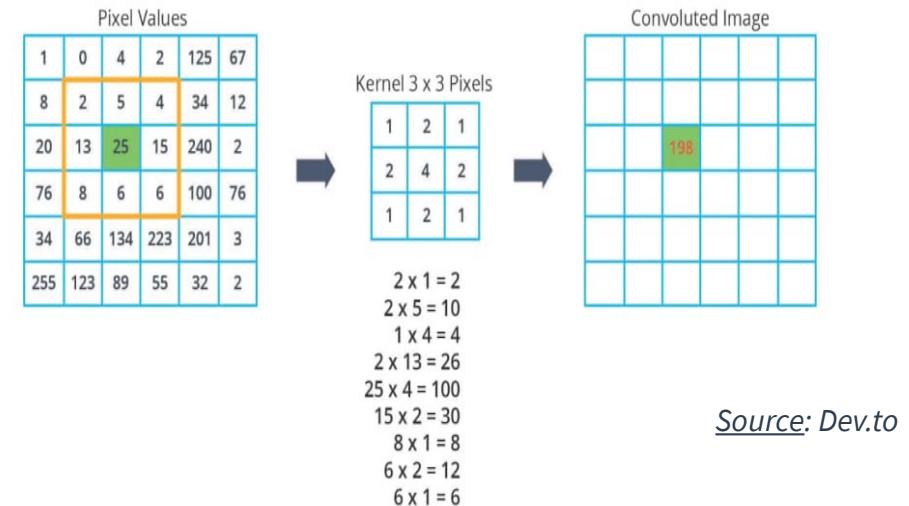
Formule de passage en noir en blanc en lien avec la luminosité :

$$pixel_gris = (pixel_rouge * 0.2989) + (pixel_vert * 0.5870) + (pixel_bleu * 0.1140)$$



1. Traitement de l'image : Initialisation de la convolution

Convolution de l'image :



Source: Dev.to

Nécessité d'un padding (noir = 0) :



1. Traitement de l'image : Filtre Gaussien

Limitation du bruit (pixels isolés) de l'image via une application du filtre gaussien :

- Calcul du noyau(kernel) de dimension 5*5 pixels avec sigma = 1.4
- Normalisation de chaque pixel
- Application de la convolution avec ce noyau

$$G(x, y) = \frac{1}{2 * \pi * \sigma^2} * e^{-\frac{x^2 + y^2}{2 * \sigma^2}}$$



1. Traitement de l'image : Filtre Sobel

Calcul du gradient de luminosité de l'image (selon x et y) via l'opérateur de dérivation de Sobel :

kernel_x :

-1	0	1
-2	0	2
-1	0	1



kernel_y :

1	2	1
0	0	0
-1	-2	-1



1. Traitement de l'image : Direction et module des gradients

Module : $|G| = \sqrt{G_x^2 + G_y^2}$

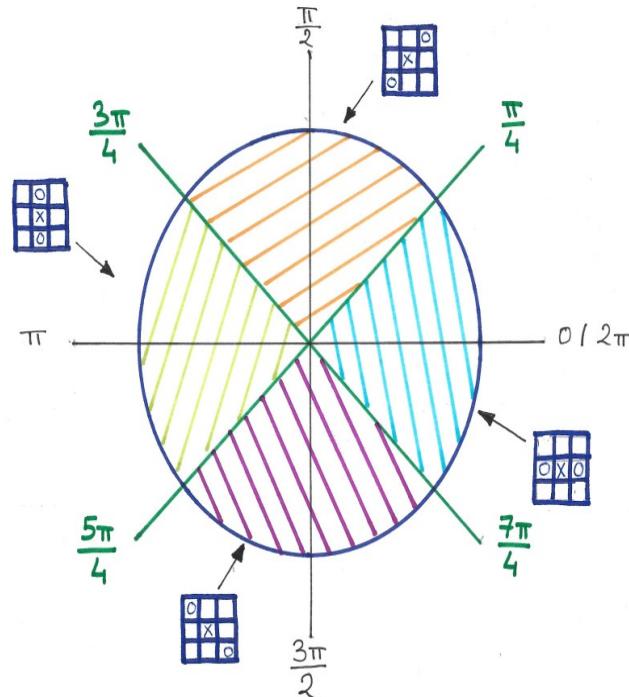


Direction : $d = \arctan(Gy/Gx)$



1. Traitement de l'image : Suppression des non-maxima

Suppression des non maxima de l'image du module via les angles contenus dans l'image direction

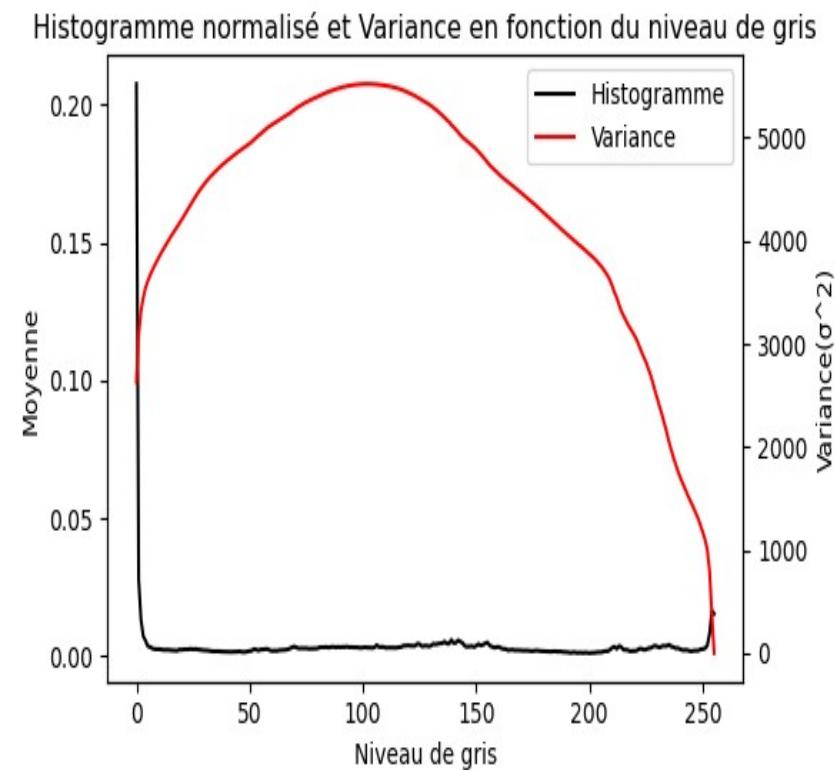


1.Traitemet de l'image : Recherche du seuil

Application de l'algorithme d'Otsu pour trouver le meilleur seuil :

- Calcul de l'histogramme et des probabilités de chaque niveau d'intensité
- Définition de $w_i(0)$ et $\mu_i(0)$
- Parcours tous les seuils possibles $t=1 \dots$ intensité max
 - Mise à jour des w_i et μ_i
 - Calculer : $\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = w_1(t)w_2(t)[\mu_1(t) - \mu_2(t)]^2$
- Le seuil désiré correspond alors au maximum

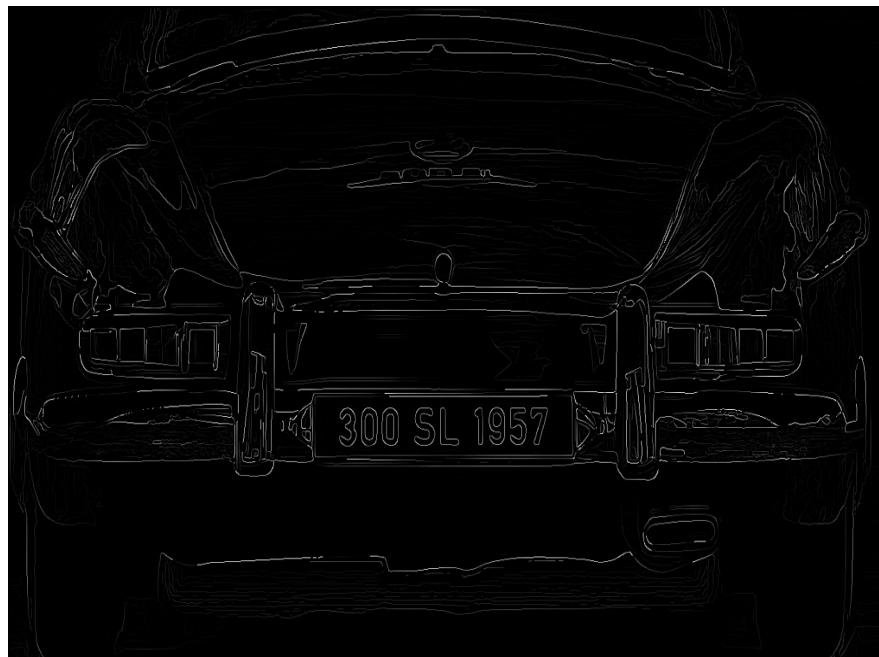
Avec w_i la probabilité de la classe i et μ_i sa moyenne



Dans notre cas le seuil vaut 102

1. Traitement de l'image : Double seuillage

Application d'un filtre de double seuillage avec : $seuil\ bas = 1/2 * seuil\ haut$



1. Traitement de l'image : Finalisation

Gestion des pixels à potentiel de contour (gris) :

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 50 & 0 \\ \hline 0 & 50 & 0 \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 50 & 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 50 & 0 \\ \hline 0 & 255 & 0 \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 255 & 0 \\ \hline 0 & 255 & 0 \\ \hline \end{array}$$



2. Extraction de la plaque

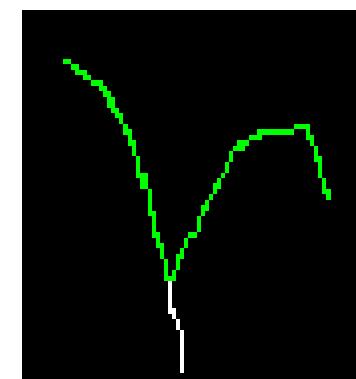
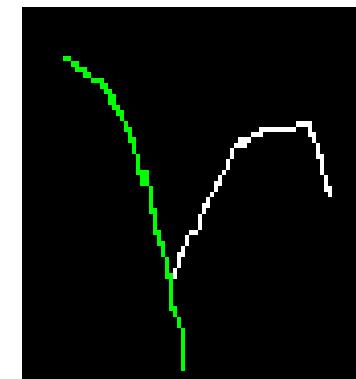
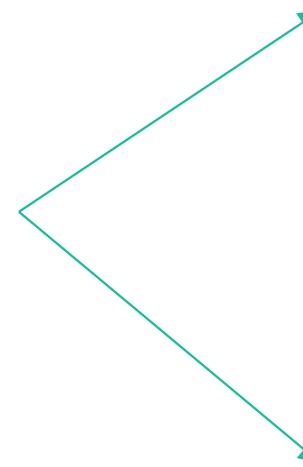
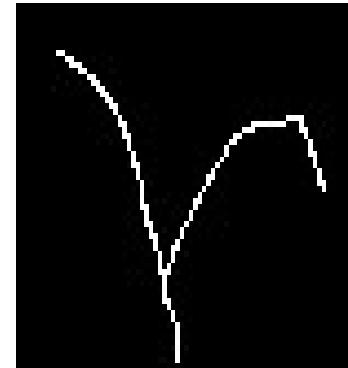
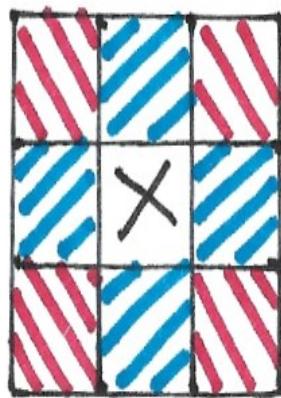
Objectif : Localisation et extraction de la plaque via les contours



300 SL 1957

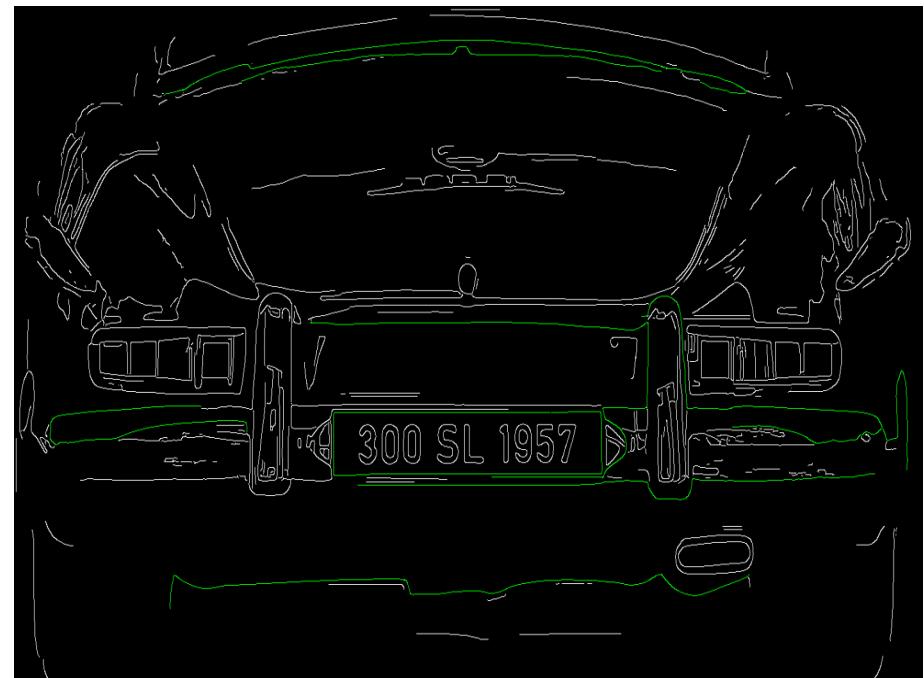
2. Extraction de la plaque: Obtention des contours

Création des contours via une recherche de proche en proche en proche de pixels blancs :



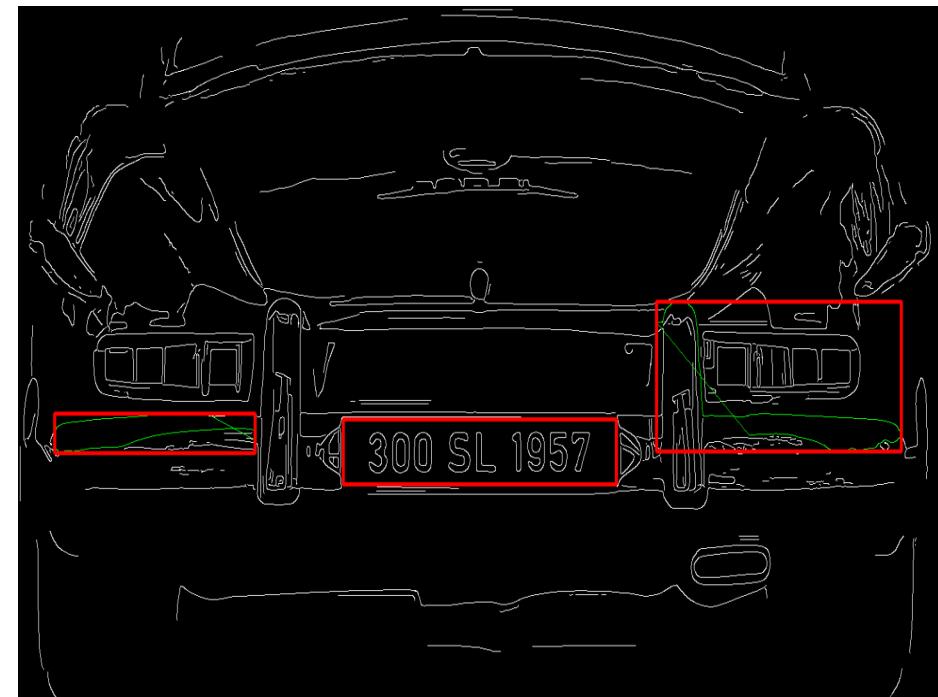
2. Extraction de la plaque: Tri des contours

Tri des contours et conservation des 20 plus longs :



2. Extraction de la plaque: Bounding box

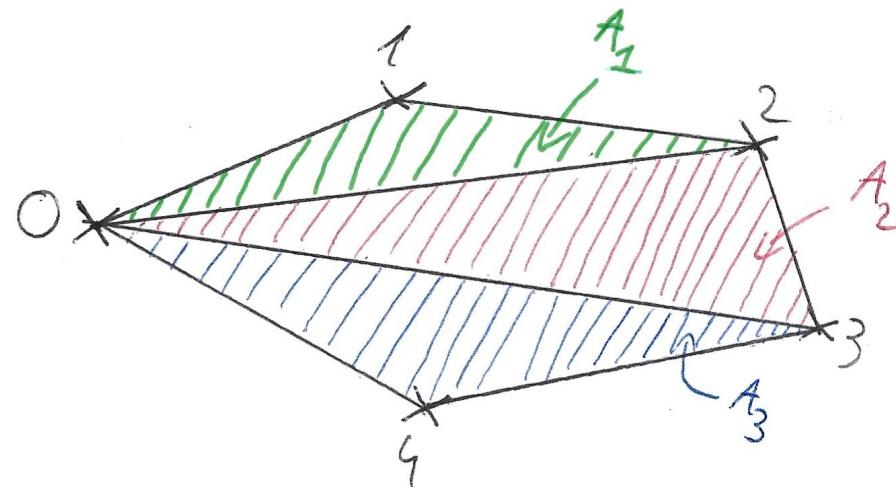
Calcul des rectangles englobants(*bounding boxes*) avec des contours considérés comme fermés :



2. Extraction de la plaque: Assimilation à un rectangle

On cherche ici à savoir si le contour fermé est assimilable à sa *bounding box*:

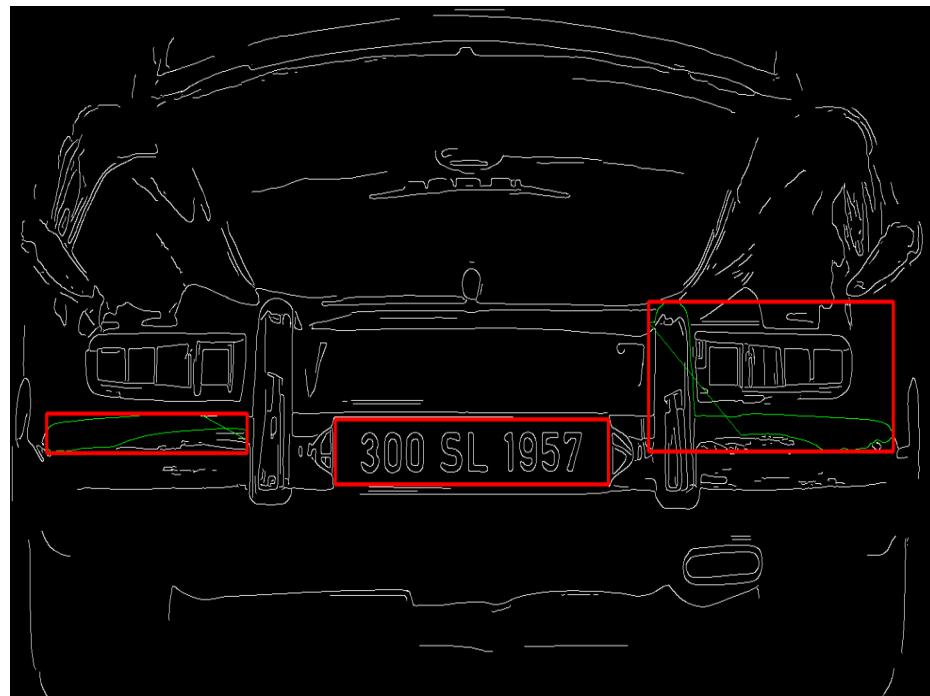
Aire d'un triangle entre 3 points $(0,0)$, (x_1,y_1) et (x_2,y_2) : $A = x_1 * y_2 - x_2 * y_1$



$$A = A_1 + A_2 + A_3$$

2. Extraction de la plaque: Vérification des proportions

Une plaque d'immatriculation fait $52 * 11$ cm, on vérifie donc que le rectangle englobant (*bounding box*) respecte bien cela (avec une tolérance de 20 %) :



300 SL 1957

3. Identification des caractères

Objectif : Reconnaissance des caractères de la plaque

300 SL 1957



300 SL 1957

3. Identification des caractères: Extraction des lettres

Mise en forme et séparation des caractères composants la plaque d'immatriculation

300 SL 1957



3 0 0 S L 1 9 5 7

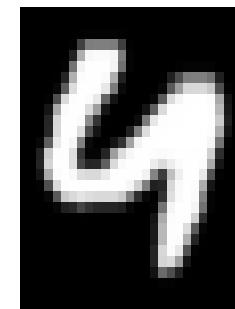
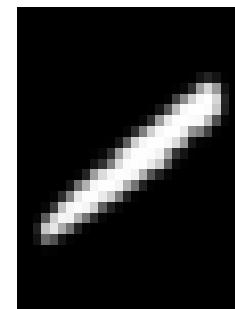
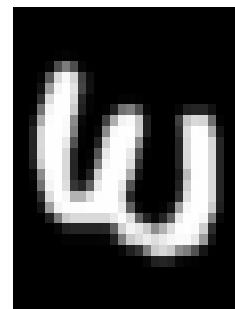
3. Identification des caractères: Importation de EMNIST

Utilisation du dataset EMNIST (Extended MNIST) pour réaliser une reconnaissance des éléments de la plaque :

Dimension: 28*28 pixels

Train : 104 800 caractères

Test : 26 200 caractères



3

1

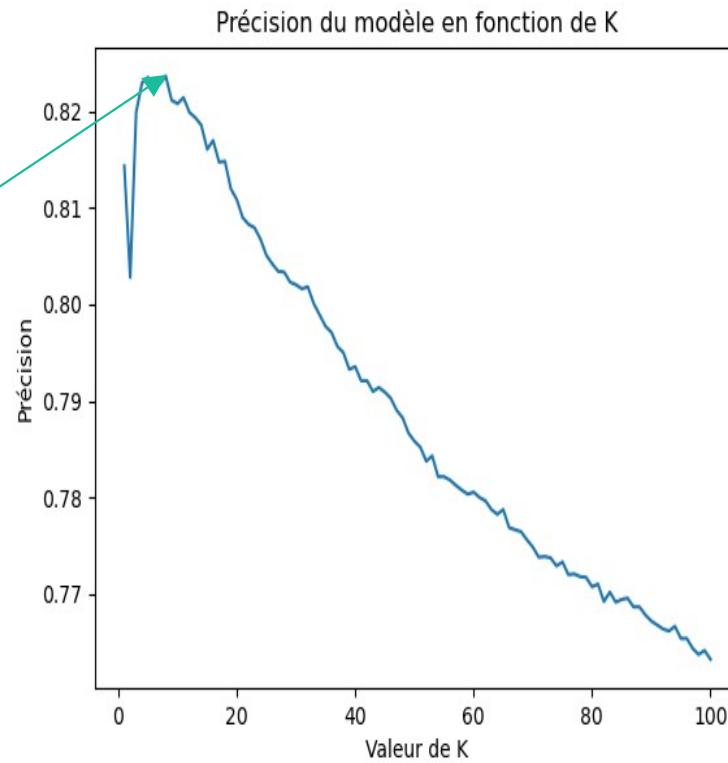
2

A

3. Identification des caractères: Algorithme K-nn

Recherche du facteur k optimal pour une reconnaissance des données :

Valeur max pour K = 8



3. Identification des caractères: Obtention des caractères

Obtention des caractères représenté sur la plaque :

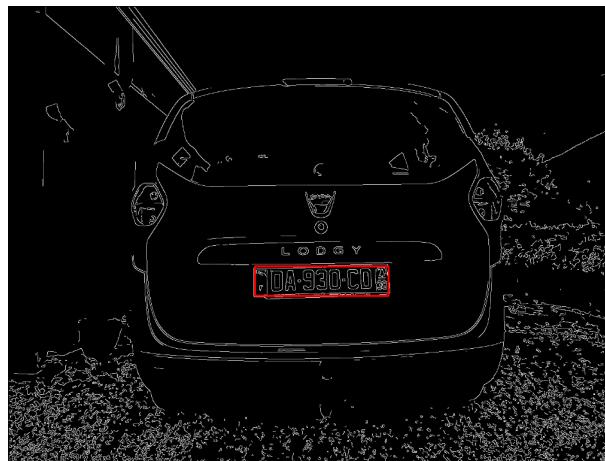
300 SL 1957



300SL1857

Conclusions : Limites

Limites du système avec une plaque sale et de nuit:



DA930EU



0A93BLU

Conclusions : Performances

Liste des différents temps d'exécution :

Filtre Canny	Méthode d'Otsu	Extraction de la plaque	Importation du modèle	Reconnaissance
70.4 s	1.1 s	68.8 s	101.1 s	8.0 s

Annexe : Filtre Canny

```
1 import numpy as np
2 import cv2
3
4 def convolve(image, kernel):
5     height, width = image.shape
6     k_height, k_width = kernel.shape
7     padding = k_width // 2 # Calcul du padding pour maintenir la taille de l'image
8
9     # Création d'une copie de l'image avec un padding
10    padded_image = np.pad(image, padding, mode='constant')
11
12    # Création de l'image résultante de la convolution
13    convolved_image = np.zeros_like(image)
14
15    # Parcours de chaque pixel de l'image
16    for i in range(height):
17        for j in range(width):
18            # Extraction de la région d'intérêt autour du pixel
19            region = padded_image[i:i + k_height, j:j + k_width]
20
21            # Application du noyau en effectuant une multiplication élément par élément
22            result = np.sum(region * kernel)
23
24            # Stockage du résultat dans l'image convoluée
25            convolved_image[i, j] = result
26
27    return convolved_image
28
29 def grayscale(image):
30     # Conversion de l'image en niveaux de gris
31     return np.dot(image[...,:3], [0.2989, 0.5870, 0.1140])
32     # Formule calcul pixel gris (en lien avec la luminosité visible d'une couleur) : pixel_gris = (pixel_rouge * 0.2989) +
33     # (pixel_vert * 0.5870) + (pixel_bleu * 0.1140)
34
35 def gaussian_blur(image, kernel_size=5, sigma=1.4):
36     # Création du noyau gaussien
37     kernel = np.fromfunction(lambda x, y: (1/(2*np.pi*sigma**2)) * np.exp(-((x-(kernel_size-1)//2)**2 + (y-(kernel_size-1)//2)**2)/(2*sigma**2)), (kernel_size, kernel_size))
38
39     # Normalisation du noyau
40     kernel /= np.sum(kernel) # On divise tout les éléments par la somme des éléments , permet de maintenir la forme de la
41     # distribution gaussienne (luminance, niveau de gris, ...)
42
43     # Application du flou gaussien en utilisant une convolution
44     blurred = convolve(image, kernel)
45
46     return blurred
```

Annexe : Filtre Canny

```
45 def sobel_filters(image):
46     # Noyaux de Sobel pour la détection des gradients horizontaux et verticaux
47     kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
48     kernel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
49
50     # Application des filtres de Sobel pour calculer les gradients horizontaux et verticaux
51     gradient_x = convolve(image, kernel_x) #Difference des intensités horizontales
52     gradient_y = convolve(image, kernel_y) #Difference des intensités verticales
53
54     return gradient_x, gradient_y
55
56 def gradient_magnitude(gradient_x, gradient_y):
57     # Calcul du module du gradient
58     magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
59
60     return magnitude
61
62 def gradient_direction(gradient_x, gradient_y):
63     # Calcul de la direction du gradient
64     direction = np.arctan2(gradient_y, gradient_x)
65
66     return direction
67
68 def non_maximum_suppression(magnitude, direction):
69     # Ajustement de la direction du gradient à des valeurs entre 0 et 180 degrés
70     direction = np.rad2deg(direction) % 180
71
72     # Dimensions de l'image
73     height, width = magnitude.shape
74
75     # Création d'une image vide pour stocker les contours supprimés
76     suppressed = np.zeros_like(magnitude)
77
78     for i in range(1, height-1):
79         for j in range(1, width-1):
80             angle = direction[i, j]
81
82             # Comparaison des magnitudes voisines selon l'orientation du gradient
83             if (0 <= angle < 22.5) or (157.5 <= angle < 180):
84                 if (magnitude[i, j] >= magnitude[i, j-1]) and (magnitude[i, j] >= magnitude[i, j+1]):
85                     suppressed[i, j] = magnitude[i, j]
86             elif (22.5 <= angle < 67.5):
87                 if (magnitude[i, j] >= magnitude[i-1, j+1]) and (magnitude[i, j] >= magnitude[i+1, j-1]):
88                     suppressed[i, j] = magnitude[i, j]
89             elif (67.5 <= angle < 112.5):
90                 if (magnitude[i, j] >= magnitude[i-1, j]) and (magnitude[i, j] >= magnitude[i+1, j]):
91                     suppressed[i, j] = magnitude[i, j]
92             elif (112.5 <= angle < 157.5):
93                 if (magnitude[i, j] >= magnitude[i-1, j-1]) and (magnitude[i, j] >= magnitude[i+1, j+1]):
94                     suppressed[i, j] = magnitude[i, j]
95
96     return suppressed
97
```

Annexe : Filtre Canny

```
98 def double_threshold(image, low_threshold, high_threshold):
99     # Dimensions de l'image
100    height, width = image.shape
101
102    # Calcul des seuils minimaux et maximaux
103    high = high_threshold
104    low = low_threshold
105
106    # Création d'une image vide pour stocker les contours après seuillage
107    thresholded = np.zeros_like(image)
108
109    # Parcours de l'image pour appliquer le seuillage
110    for i in range(height):
111        for j in range(width):
112            pixel = image[i, j]
113
114            # Application du seuil haut
115            if pixel >= high:
116                thresholded[i, j] = 255
117            # Application du seuil bas
118            elif pixel >= low:
119                thresholded[i, j] = 50 # 50 représente les pixels à potentiel de contour
120            # Pixels en dessous du seuil bas
121            else:
122                thresholded[i, j] = 0
123
124    return thresholded
125
126 def edge_tracking(image):
127     # Dimensions de l'image
128     height, width = image.shape
129
130     # Parcours de l'image pour effectuer le suivi des contours
131     for i in range(1, height-1):
132         for j in range(1, width-1):
133             # Si un pixel à potentiel de contour est détecté
134             if image[i, j] == 50:
135                 # Recherche des pixels voisins qui ont également un potentiel de contour
136                 neighbors = image[i-1:i+2, j-1:j+2] #Cerclz 3*3 autour
137                 if 255 in neighbors:
138                     image[i, j] = 255
139                 else:
140                     image[i, j] = 0
141
142     return image
143
```

Annexe : Filtre Canny

```
144 # Chargement de l'image
145 image = cv2.imread('voiture4.png')
146
147 # Conversion en niveaux de gris
148 gray = grayscale(image)
149
150 # Application du flou gaussien
151 blurred = gaussian_blur(gray, kernel_size=5, sigma=1.4)
152
153 # Calcul des gradients avec les filtres de Sobel
154 gradient_x, gradient_y = sobel_filters(blurred)
155
156 # Calcul du module du gradient
157 magnitude = gradient_magnitude(gradient_x, gradient_y)
158
159 # Calcul de la direction du gradient
160 direction = gradient_direction(gradient_x, gradient_y)
161
162 # Suppression des pixels non-maximaux
163 suppressed = non_maximum_suppression(magnitude, direction)
164
165 # Seuillage double
166 thresholded = double_threshold(suppressed, low_threshold=51, high_threshold=102)
167
168 # Suivi des contours
169 edges = edge_tracking(thresholded)
170
171 # Affichage de l'image avec les contours détectés
172 edges = np.uint8(edges)
173 cv2.imshow('Edges', edges)
174 cv2.waitKey(0)
175 cv2.destroyAllWindows()
```

Annexe : Algorithme d’Otsu

```
1 import numpy as np
2 import cv2
3 import matplotlib.pyplot as plt
4
5 def find_canny_threshold(image):
6     # Conversion de l'image en niveaux de gris
7     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
8
9     # Calcul de l'histogramme normalisé
10    hist = cv2.calcHist([gray], [0], None, [256], [0, 256])
11
12    hist_norm = hist.ravel() / hist.sum() # Méthode ravel applatit l'histogramme (tableau à 1 dimension)
13
14    # Calcul de la somme cumulée normalisée
15    cumsum = np.cumsum(hist_norm)
16
17    # Calcul de la somme cumulée moyenne
18    cumsum_mean = np.cumsum(hist_norm * np.arange(256))
19
20    threshold = 0
21    var_max = 0
22    sigma = []
23
24    # Recherche du seuil optimal
25    for t in range(256):
26        p1 = cumsum[t] # Proba que le pixel t
27        p2 = 1 - p1
28
29        # Si un des pixels voisins vaut 0 (noir)
30        if p1 == 0 or p2 == 0: # On saute l'itération si la condition est vérifiée
31            continue
32
33        mean1 = cumsum_mean[t] / p1
34        mean2 = (cumsum_mean[-1] - cumsum_mean[t]) / p2
35
36        var = p1 * p2 * (mean1 - mean2)**2 # Formule de l'algorithme d'Otsu
37        sigma.append(var)
38
39        if var > var_max:
40            var_max = var
41            threshold = t
42
43    print(threshold)
44
45    # Configuration de la figure avec deux axes Y
46    fig, ax1 = plt.subplots(figsize=(6, 4))
47    ax2 = ax1.twinx()
48
```

Annexe : Algorithme d'Otsu

```
49 # Tracé de l'histogramme normalisé
50 ax1.plot(hist_norm, color='black', label='Histogramme')
51 ax1.set_xlabel('Niveau de gris')
52 ax1.set_ylabel('Moyenne')
53 ax1.set_title('Histogramme normalisé et Variance en fonction du niveau de gris')
54
55 # Tracé de la variance
56 ax2.plot(sigma, color='red', label='Variance')
57 ax2.set_ylabel('Variance( $\sigma^2$ )')
58
59 # Positionnement de la légende en haut à droite
60 lines, labels = ax1.get_legend_handles_labels()
61 lines2, labels2 = ax2.get_legend_handles_labels()
62 ax2.legend(lines + lines2, labels + labels2, loc='upper right', bbox_to_anchor=(1.0, 1.0))
63
64 plt.tight_layout()
65 plt.show()
66
67 # Charger l'image
68 image = cv2.imread('voiture4.png')
69 find_canny_threshold(image)
```

Annexe : Détection des contours

```
1 import cv2
2 import numpy as np
3 import sys
4
5 # Chargement de l'image
6 edges = cv2.imread('Eges4.png', 0)
7
8 #Modification de la profondeur de recherche:
9 new_limit = 5000 # Nouvelle limite de profondeur maximale de récursion
10 sys.setrecursionlimit(new_limit)
11
12 #Utilitaire
13 def distance(x1,y1,x2,y2):
14     return np.sqrt((x1-x2)**2 + (y1-y2)**2)
15 #Obtention des contours
16 height, width = edges.shape[:2]
17 contours = []
18 visited = []
19
20 def complet_visited(L): #Incrementation de la list permettant de ne pas refaire plusieurs contours
21     for i in L:
22         if i not in visited:
23             visited.append(i)
24
25 def next_w_pixel(i, j, image, temp_visited, not_allowed):
26     L = []
27     mouv_adjacent = [(0, -1), (-1, 0), (0, 1), (1, 0)] # Déplacements pour les pixels adjacents (haut, gauche, bas, droite)
28     mouv_diagonal = [(-1, -1), (-1, 1), (1, -1), (1, 1)] # Déplacements pour les pixels en diagonale
29     adjacent_found = False
30
31     no_move = temp_visited + not_allowed #Ces points ne comptent pas
32
33     for mov in mouv_adjacent:
34         a, b = mov
35         if 0 <= i + a < height and 0 <= j + b < width: # Vérification pixel dans l'image
36             if image[i + a][j + b] == 255:
37                 if (i + a, j + b) not in no_move:
38                     L.append((i + a, j + b))
39                     adjacent_found = True
40
41     if not adjacent_found:
42         for mov in mouv_diagonal:
43             a, b = mov
44             if 0 <= i + a < height and 0 <= j + b < width: # Vérification pixel dans l'image
45                 if image[i + a][j + b] == 255:
46                     if (i + a, j + b) not in no_move and (i + a, j + b) not in visited:
47                         L.append((i + a, j + b))
48
49 return L
50
```

Annexe : Détection des contours

```
51 def extractContours(i, j, image, temp_visited, not_allowed):
52     temp_visited.append((i, j))
53     next = next_w_pixel(i, j, image, temp_visited, not_allowed)
54     if len(next) == 0:
55         contours.append(temp_visited)
56         complet_visited.append(temp_visited)
57
58     elif len(next) == 1:
59         extractContours(next[0][0], next[0][1], image, temp_visited, not_allowed)
60
61     else:
62         for i in range(len(next)):
63             t_visited = temp_visited.copy() #Copie du parcours commun
64             t_not_allowed = not_allowed.copy()
65             r_next = next.copy()
66             r_next.remove(next[i])
67             extractContours(next[i][0], next[i][1], image, t_visited, t_not_allowed + r_next)
68
69 def count_neighbour_pixels(i, j, image):
70     count = 0
71     height, width = image.shape
72
73     # Coordonnées des 8 pixels voisins
74     neighbours = [(i-1, j-1), (i-1, j), (i-1, j+1),
75                    (i, j-1), (i, j+1),
76                    (i+1, j-1), (i+1, j), (i+1, j+1)]
77
78     for neighbour in neighbours:
79         x, y = neighbour
80         if 0 <= x < height and 0 <= y < width: # Vérification si les coordonnées sont valides
81             if image[x][y] == 255: # Vérification si le pixel est blanc
82                 count += 1
83
84     return count
85
86 def is_extremity(i, j, image): #Extémité pour commencer la détection la plus proche
87     count = count_neighbour_pixels(i,j,image)
88     if count < 2:
89         return True
90     else :
91         return False
92
93 def detectContours(image):
94     for i in range(height):
95         for j in range(width):
96             if image[i][j] == 255 and (i, j) not in visited:
97                 if is_extremity(i, j, image):
98                     extractContours(i, j, image, [], [])
99
100 detectContours(edges)
```

Annexe : Détection des contours

```
103 def sort_and_get_top_contours(contours):
104     # Trier les contours par leur aire (taille)
105     sorted_contours = sorted(contours, key=len, reverse=True)
106
107     # Récupérer les 50 plus grands contours
108     top_contours = sorted_contours[:20]
109
110     return top_contours
111
112 print(len(contours))
113 contours = sort_and_get_top_contours(contours)
114
115 #Bounding box:
116 def bounding_box(contour):
117     # min de y (ligne)
118     min_row = min(contour, key=lambda x: x[0])[0] #key=lambda x: x[0] signifie que l'on trie selon la première coordonée (ici y)
119     # max de y
120     max_row = max(contour, key=lambda x: x[0])[0]
121
122     # min de x
123     min_col = min(contour, key=lambda x: x[1])[1]
124
125     # max de x
126     max_col = max(contour, key=lambda x: x[1])[1]
127
128     # Calcul des dimensions du rectangle englobant
129     width = max_col - min_col
130     height = max_row - min_row
131
132     return min_row, min_col, width, height # (min_row,max_row) les coords du point en haut a gauche
133
```

Annexe : Détection des contours

```
134 #Verification du fait qu'un contour soit fermé
135 def tracer_ligne(image, x1, y1, x2, y2):
136     # Créer une copie de l'image pour ne pas modifier l'originale
137     image_ligne = np.copy(image)
138
139     # Tracer la ligne blanche
140     cv2.line(image_ligne, (x1, y1), (x2, y2), (255), 1)
141
142     # Calculer les coordonnées des points intermédiaires
143     dx = abs(x2 - x1)
144     dy = abs(y2 - y1)
145     steps = max(dx, dy)
146
147     points_ligne = []
148
149     if steps > 0:
150         x_step = (x2 - x1) / steps
151         y_step = (y2 - y1) / steps
152
153         x = x1
154         y = y1
155
156         # Ajouter les points intermédiaires à la liste
157         for _ in range(steps - 1):
158             x += x_step
159             y += y_step
160             points_ligne.append((int(round(x)), int(round(y))))
161
162     return points_ligne
163
164
165 def closed_contour(image,contours):
166     c_contours = []
167     for contour in contours:
168         depart = contour[0]
169         fin = contour[-1]
170         dist = distance(depart[0],depart[1],fin[0],fin[1])
171
172         if dist < (1/4) * len(contour):
173             points_ligne = tracer_ligne(image,depart[0],depart[1],fin[0],fin[1])
174             c_contours.append(contour + points_ligne)
175
176     return c_contours
177
```

Annexe : Détection des contours

```
179 #Calcul de l'air d'un contour supposé fermé
180 def calculate_contour_area(contour):
181     area = 0
182     n = len(contour)
183
184     for i in range(n):
185         x1, y1 = contour[i]
186         x2, y2 = contour[(i + 1) % n] # Point suivant (bouclage au premier point)
187
188         area += (x1 * y2 - x2 * y1)
189
190     area = abs(area) / 2 #Ainsi l'air réel est positive
191     print (area)
192     return int(area)
193
194 #Test pour savoir si le contour est assimilable à sa box englobante (plus un contour est à la fin de la liste plus il est proche
de sa box
195 def is_rectangle(image, cContours):
196     contours_proches = []
197
198     for contour in cContours:
199         box = bounding_box(contour)
200         box_area = box[2] * box[3]
201         contour_area = calculate_contour_area(contour)
202
203         proximity = abs(box_area - contour_area)
204         contours_proches.append((contour, proximity))
205
206     contours_proches = sorted(contours_proches, key=lambda x: x[1], reverse=False) #On trie selon la proximity d'où x[1]
207     sortedContours = [contour for contour, _ in contours_proches]
208
209     return sortedContours
210
211 #Vérification du ratio
212 def is_plate(image, cContours):
213     plaque = None
214     for contour in is_rectangle(image, cContours):
215         x, y, w, h = bounding_box(contour)
216         if w > h :
217             ratio = w/h #ratio d'une plaque 52/11
218             if abs(ratio-52/11) < 0.2*(52/11) : #Seuil de tolérance de 20 pourcents
219                 plaque = contour
220
221     return plaque
222
```

Annexe : Détection des contours

```
223 #Extraction
224 def crop_image(image, x, y, w, h):
225     # Récupérer la partie de l'image correspondant au rectangle
226     cropped_image = image[y+10:y+h-10, x+10:x+w-10].copy() #On enlève 10 pixels pour ne plus avoir le contour de la plaque
227
228     # Redimensionner l'image pour correspondre exactement à la taille du rectangle
229
230     return cropped_image
231
232 #Utilisation
233 c_contours = closed_contour(edges,contours)
234
235 image_with_points = cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB)
236 for contour in c_contours:
237     for pixel in contour:
238         cv2.circle(image_with_points, (pixel[1], pixel[0]), 0, (0, 255, 0), -1) # Dessin des contours en vert
239
240     y, x, w, h = bounding_box(contour)
241     cv2.rectangle(image_with_points, (x, y), (x + w, y + h), (0, 0, 255), 3)
242
243 cv2.imshow('Image avec contours', image_with_points)
244 cv2.waitKey(0)
245 cv2.destroyAllWindows()
246
247
```

Annexe : Séparation des caractères

```
1 import numpy as np
2 import cv2
3 import os
4
5 def seuillage(image, seuil):
6     # Parcourir tous les pixels de l'image
7     for i in range(image.shape[0]):
8         for j in range(image.shape[1]):
9             # Appliquer le seuillage
10            if image[i, j] < seuil:
11                image[i, j] = 0 # Noir
12            else:
13                image[i, j] = 255 # Blanc
14
15    return image
16
17 #Initialisation de la plaque
18 image_plaque = cv2.imread('Plaque5.png', 0)
19 height, width = image_plaque.shape[:2]
20 transformed = seuillage(image_plaque, 100)
21 transformed = cv2.bitwise_not(image_plaque) # Inversion du noir et blanc (pour utilisation dans la fonction)
22
23 #Decoupage de la plaque
24 def decoup_plaque (image):
25     detect = False
26     x_start,x_end = None,None
27     L = []
28     for i in range(width):
29         n_line = 0
30         for j in range(height):
31             if image[j][i] == 255:
32                 if detect == False: #Est ce qu'un chiffre est déjà en train d'être détecté
33                     detect = True
34                     x_start = i
35                     x_end = i
36
37                 else :
38                     if i < x_start :
39                         x_start = i
40                     if i > x_end :
41                         x_end = i
42
43             elif detect == True:
44                 n_line += 1
45
46             if n_line == height: #Fin de collone
47                 if x_start != None:
48                     detect = False
49                     if x_start != x_end: #Pour éviter les pixels seuls
50                         L.append([x_start,x_end])
51
52
53 return L
```

Annexe : Séparation des caractères

```
52
53 def nettoyage():
54     folder_path = "lettres"
55     # Parcourir tous les fichiers et dossiers dans le dossier
56     for filename in os.listdir(folder_path):
57         file_path = os.path.join(folder_path, filename)
58         os.remove(file_path)
59
60
61 def crop_image(image):
62     L = decoup_plaque(image)
63     nettoyage()
64     for i in range(len(L)):
65         lettre = L[i]
66         cropped_image = image_plaque[0:height, lettre[0]:lettre[1]].copy()
67
68         #Enregistrement
69         letter_path = os.path.join("lettres", f"{i}.png")
70         cv2.imwrite(letter_path, cropped_image)
71
72
73 crop_image(transformed)
74
75
```

Annexe : Valeur de K

```
1 import numpy as np
2 import cv2
3 import pandas as pd
4 from sklearn.model_selection import train_test_split
5 from sklearn.neighbors import KNeighborsClassifier
6 import matplotlib.pyplot as plt
7
8 # Chargement des données EMNIST Balanced Train
9 train_data = pd.read_csv('emnist-balanced-train.csv')
10
11 # Filtre pour inclure uniquement les lettres majuscules dans l'ensemble de train
12 train_letters_mask = (train_data.iloc[:, 0] >= 10) & (train_data.iloc[:, 0] <= 35) | (train_data.iloc[:, 0] < 10)
13 train_data = train_data.loc[train_letters_mask]
14
15 # Chargement des données EMNIST Balanced Test
16 test_data = pd.read_csv('emnist-balanced-test.csv')
17
18 # Filtre pour inclure uniquement les lettres majuscules dans l'ensemble de test
19 test_letters_mask = (test_data.iloc[:, 0] >= 10) & (test_data.iloc[:, 0] <= 35) | (test_data.iloc[:, 0] < 10)
20 test_data = test_data.loc[test_letters_mask]
21
22 # Séparation des caractéristiques (pixels) et des étiquettes (lettres ou chiffres) pour l'ensemble de train
23 X_train = train_data.iloc[:, 1:].values
24 y_train = train_data.iloc[:, 0].values
25
26 # Séparation des caractéristiques (pixels) et des étiquettes (lettres ou chiffres) pour l'ensemble de test
27 X_test = test_data.iloc[:, 1:].values
28 y_test = test_data.iloc[:, 0].values
29
30 # Listes pour stocker les valeurs de K et les précisions correspondantes
31 k_values = []
32 accuracies = []
```

Annexe : Valeur de K

```
34 # Boucle sur les valeurs de K de 1 à 100
35 for k in range(1, 101):
36     # Création du classifieur k-nn
37     knn = KNeighborsClassifier(n_neighbors=k)
38
39     # Entraînement du classifieur
40     knn.fit(X_train, y_train)
41
42     # Évaluation des performances du modèle sur l'ensemble de test
43     accuracy = knn.score(X_test, y_test)
44
45     # Ajout des valeurs de K et de la précision à leurs listes respectives
46     k_values.append(k)
47     accuracies.append(accuracy)
48
49     print("K =", k, "- Exactitude du modèle :", accuracy)
50
51 # Tracé de la précision en fonction de K
52 plt.plot(k_values, accuracies)
53 plt.xlabel('Valeur de K')
54 plt.ylabel('Précision')
55 plt.title('Précision du modèle en fonction de K')
56 plt.show()
57
```

Annexe : Reconnaissance des caractères

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from PIL import Image
4 import pandas as pd # Bibliotheque de lecture de tableaux (csv)
5 from sklearn.model_selection import train_test_split
6 from sklearn.neighbors import KNeighborsClassifier
7 import os
8 from pathlib import Path
9
10 BASE_DIR = Path(__file__).resolve().parent
11 file_csv = os.path.join(BASE_DIR, "emnist-balanced-train.csv")
12
13 print("Chargement des données EMNIST Balanced Train")
14 train_data = pd.read_csv(file_csv)
15
16 print("Filtre pour inclure uniquement les lettres majuscules dans l'ensemble de train")
17
18 train_letters_mask = (train_data.iloc[:, 0] >= 10) & (train_data.iloc[:, 0] <= 35) | (train_data.iloc[:, 0] < 10)
19 train_data = train_data.loc[train_letters_mask]
20
21
22
23 print("Chargement des données EMNIST Balanced Test")
24 file_csv = os.path.join(BASE_DIR, "emnist-balanced-test.csv")
25 test_data = pd.read_csv(file_csv)
26
27
28
29 print("Filtre pour inclure uniquement les lettres majuscules dans l'ensemble de test")
30 test_letters_mask = (test_data.iloc[:, 0] >= 10) & (test_data.iloc[:, 0] <= 35) | (test_data.iloc[:, 0] < 10)
31 test_data = test_data.loc[test_letters_mask]
32
33
34 print("Séparation des caractéristiques (pixels) et des étiquettes (lettres ou chiffres) pour l'ensemble de train")
35
36 X_train = train_data.iloc[:, 1:].values
37 y_train = train_data.iloc[:, 0].values
38
39 print("Séparation des caractéristiques (pixels) et des étiquettes (lettres ou chiffres) pour l'ensemble de test")
40
41 X_test = test_data.iloc[:, 1:].values
42 y_test = test_data.iloc[:, 0].values
43
44 print("Création du classifieur k-nn")
45
46 knn = KNeighborsClassifier(n_neighbors=8)
47
48 print("Entrainement du classifieur")
49
50 knn.fit(X_train, y_train)
51
52 print("Évaluation des performances du modèle sur l'ensemble de test")
```

Annexe : Reconnaissance des caractères

```
53 accuracy = knn.score(X_test, y_test)
54
55 print("Exactitude du modèle : ", accuracy)
56
57 # Fonction pour prédire une lettre ou un chiffre à partir d'une image
58
59 def predict_symbol(image):
60     image = np.array(image).reshape(1, -1)
61     predicted_symbol = knn.predict(image)
62     print(predicted_symbol)
63
64     if predicted_symbol[0] < 10:
65         # Renvoyer un chiffre en tant que chaîne de caractères
66         return str(predicted_symbol[0])
67
68     else:
69         # Renvoyer une lettre majuscule en tant que caractère
70         return chr(predicted_symbol[0] + 55)
71
72 # Transformation d'une image en liste utilisable dans predict_symbol
73
74 def image_normalized(image_path):
75     # Convertir l'image en niveaux de gris
76     image = Image.open(image_path).convert("L")
77
78     # Inversion du noir et blanc (pour utilisation dans la fct)
79     image = image.point(lambda pixel: 255 - pixel)
80
81     # Redimensionner l'image à la taille attendue (28 pixels max)
82     h, w = image.size
83     print("taille initiale", (h, w))
84     maxi = max(h, w)
85     im = None
86
87     if abs(h-w) < (3/4)*maxi:
88         h, w = int(28 * h / maxi), int(28 * w / maxi)
89         print(h,w)
90         print("taille max 28", (h, w))
91         resized_image = image.resize((h, w))
92
93         # Vignette centrée sur carré 28x28 pixels
94         im = Image.new("RGB", (28, 28))
95         im.paste(resized_image, box=((28 - h) // 2, (28 - w) // 2))
96
97         # Orientation pour analyse knn
98
99         im = im.rotate(90).transpose(Image.Transpose.FLIP_TOP_BOTTOM)
100
101 return im
```

Annexe : Reconnaissance des caractères

```
102
103 def img_to_list(img):
104     arr = np.array(img.getdata())
105     pixel_values = list(arr[:, 0])
106
107     return pixel_values
108
109 print("Exemple d'utilisation")
110
111 # Récupérer la liste des fichiers dans le dossier "lettres"
112 lettres_folder = os.path.join(BASE_DIR, "lettres")
113 image_files = os.listdir(lettres_folder)
114
115 for file_name in image_files:
116     file_png = os.path.join(lettres_folder, file_name)
117     image_to_predict = image_normalized(file_png)
118     if image_to_predict != None:
119         pixel_values = img_to_list(image_to_predict)
120         predicted_symbol = predict_symbol(pixel_values)
121         print("Fichier :", file_name)
122         print("Symbole prédit :", predicted_symbol)
123
124
```