RAPPORT DE PROJET

Rapport n°2 - Signal

Dimanche 26 Avril 2020

CHAN Jayvin, GOOR Aurélien GE4

RECONNAISSANCE OPTIQUE DE CARACTERES

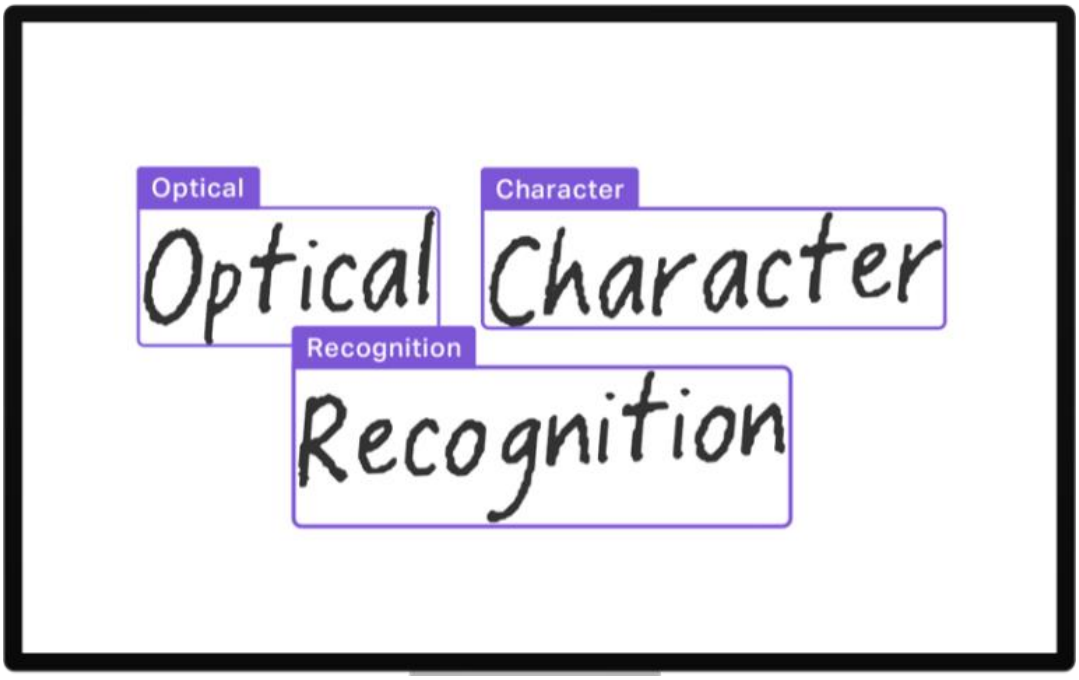

Table des matières

I. Etat de l'art

La reconnaissance optique de caractères consiste à générer un texte à partir d'une image contenant un texte.

De nos jours, de nombreux logiciels performants existent déjà. Chez les particuliers avec l'utilisation sur smartphone ou dans l'industrie avec la reconnaissance des textes sur les objets, cet outil s'avère très utile. Il existe trois grandes techniques de reconnaissance :

*les méthodes statistiques dans le domaine de la reconnaissance d'écriture manuscrite

*la classification par caractéristiques (Features) basée sur la reconnaissance des formes

*les méthodes métriques consistant à comparer la forme à reconnaître à un ensemble de modèles

Dans le cadre du mini-projet de signal, nous nous proposons donc d'implémenter une reconnaissance optique de caractères à partir des notions vues en cours, telles que l'intercorrélation. Il faudrait donc repérer chaque caractère présent sur l'image, et identifier chaque caractère en le comparant à une base de données.

Nous avons commencé par faire des recherches pour avoir des grandes lignes pour la réalisation de notre projet de reconnaissance optique de caractère.
Nous nous sommes inspirés du site suivant :
https://moov.ai/fr/blog/reconnaissance-optique-de-caracteres-ocr/

Afin de reconnaître les caractères de l'image, nous procédons de la manière suivante :

- isolation du caractère
- comparaison à tous les caractères de la base de données (voir partie correspondante, calcul non effectué si de base il y a une grande différence)
- écriture des caractères reconnus

II. Prétraitement

1. Base de données

Dans un premier temps nous réalisons une base de données contenant les différents caractères à comparer. Nous utilisons la police monospace `'Courier New'`, chaque caractère est transformé en image comme vu ci-dessous :

Figure 1 - Caractère en image

Script

```
%------------------------------------------------------------------------
%Image de reference
majuscule = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'];
minuscule = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s',
't','u','v','w','x','y','z'];
chiffre = ['0','1','2','3','4','5','6','7','8','9'];
Liste_Caractere = [majuscule , minuscule , chiffre];
%Importation dans la Base de donnees
BDD= importation(Liste_Caractere);
```

Nous réalisons une fonction capable d'automatiser l'importation des images de chaque caractère et de les stocker dans une base de données sous forme de cellule (cell).

```
function bdd = importation(Liste_Caractere)
 N = length (Liste_Caractere);

bdd = cell (1,N);

    %Importation des caracteres majuscules
    for i = 1:26
        nom = strcat('01 - base de donnees/',Liste_Caractere(i),'_monospace.PNG');
        bdd{i} = imcomplement(rgb2gray(imread(nom)));%imcomplement pour l'inversion noir/blanc
    end

    %Importation des caracteres minuscules
    for i = 27:52
        nom = strcat('01 - base de donnees/',Liste_Caractere(i),'_petit.PNG');
        bdd{i} = imcomplement(rgb2gray(imread(nom)));
    end

    %Importation des chiffres
    for i = 53:62
        nom = strcat('01 - base de donnees/',Liste_Caractere(i),'.PNG');
        bdd{i} = imcomplement(rgb2gray(imread(nom)));
    end
end
```

2. Traitement de l'image

Après avoir importé l'image contenant le texte à détecter, le premier traitement à effectuer est de convertir l'image en niveau de gris et de corriger l'inclinaison si nécessaire.

2.1. Conversion en niveau de gris

Pour convertir une image en couleur en niveaux de gris nous calculons la luminance de chaque pixel de la façon suivante :

$$L = 0.299R + 0.587V + 0.114B$$

La commande de Matlab 'rgb2gray' permet de faire la conversion en niveaux de gris. Nous l'avions utilisée au départ, mais il le programme de reconnaissance fonctionne mieux lorsque « le noir est vraiment noir et le blanc vraiment blanc » c'est pourquoi nous utilisons 'im2bw' afin d'obtenir une séparation nette des zones noires et blanches. Cela est fait de la manière suivante :

```
img = uint8(im2bw(img)*255);
```

2.2. Rotation de l'image

Pour que le programme de reconnaissance fonctionne au mieux, il est judicieux de s'assurer que les lignes soient horizontales.

Pour bien comprendre comment fonctionne ce programme on peut faire une métaphore avec une ombre projetée. Si l'on tient une feuille sous une lampe sont ombre sera minimale lorsque la feuille sera verticale.

De même si pour plusieurs angles de rotation (ici entre -5° et +5°) nous comptons le nombre de ligne ne comptant aucun pixel noir alors le nombre de ligne blanche sera maximal lorsque le texte sera horizontal.

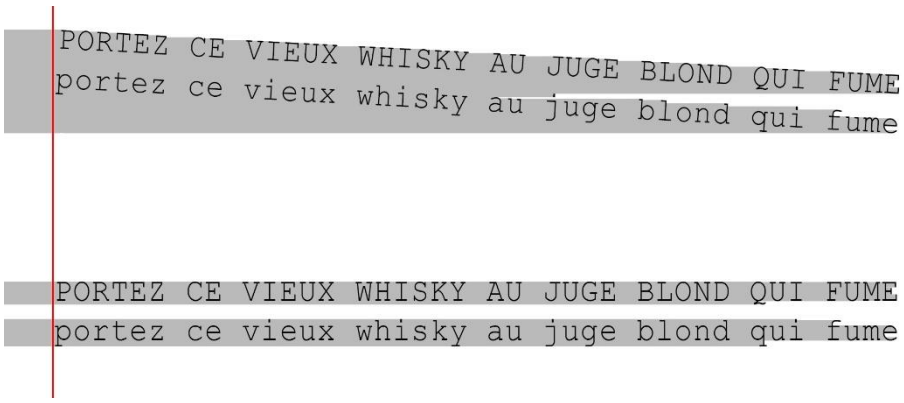

Figure 2 : Illustration ombre du texte

Nous créons alors une fonction de traitement qui prend une image en entrée et qui retourne une image 'droite' et en niveaux de gris.

Script

```
function nouvelle_image = traitement ( img )
    %Conversion en niveau de gris
    disp('Conversion en niveau de gris');
    img = uint8(im2bw(img)*255);

    %Detection de la rotation de l'image
    disp('Detection de la rotation de l image');
    img = imcomplement(img);       %Imcomplement permet d'inverser le blanc et le noir, en
                                    effet lors de la rotation , des parties noires sont
                                    complétées afin de garder le format rectangulaire droit
    angle = 0;
    max = 0;
    for theta = -5.0:0.1:5.0       %Angle de rotation entre -5 et -5 degres
        img2 = imrotate(img,theta);
        vecteur = sum(img2'>128);
        nulles = sum ( vecteur < 5);    %nombre de lignes sans pixel noir qui est maximal
                                         lorsque l'image est "droite"
        if nulles > max
            max = nulles;
            angle = theta;
        end
    end
    nouvelle_image = imrotate (img , angle);
    nouvelle_image = imcomplement(nouvelle_image);     %On complémente de nouveau pour se
                                                        ramener à du noir sur du blanc
    disp('Angle');
    disp(angle);
end
```

III. Segmentation

Le principe est de délimiter chaque ligne de l'image, et pour chaque ligne d'isoler chaque caractère pour la détection.

1. Détection des lignes

L'étape suivante est la détection des lignes de texte, pour ce faire on encadre cette ligne de texte par deux lignes horizontales virtuelles comme illustré ci-dessous :

Figure 3 : Détection des lignes de texte

Au préalable nous définissons un seuil qui vaut par exemple 128 qui permettra de définir si un pixel est noir ou blanc à partir du niveau de gris.

Script

```
    %Detection de la ligne haute
    l = yb+1;
    existe = 0;                                 %0 s'il n'y a aucun pixel noir sur la ligne

    while (not (existe)) && (l<hauteur)         %Recherche d'une ligne contenant un pixel noir
        existe = sum(img(l, :)<= seuil) >0;     %Detection d'un pixel noir sur la ligne l
        l=l+1;
    end
    yh=l-1;
```

```
%Detection de la ligne basse
    existe = 1;                                 %Existe t-il au moins un noir
    while (existe) && (l<hauteur)               %Recherche d'une ligne entierement blanche
        existe = sum(img(l, :)<= seuil) >0;     %Si il y a au moins un pixel noir
        l=l+1;
    end
    yb=l-1;
```

2. Isolation des caractères

On procède de la même façon que pour la détection des lignes haute et basse, mais de façon verticale :

Figure 4 : Détections des lettres

Script

```
%-------------------------------------------------
%Detection du début du premier caractère
existe = 0;                                 %0 s'il n'y a aucun pixel noir sur la colonne
c=fin+1;
while (not (existe)) && (c<longueur)        %Recherche d'une colonne avec au moins un pixel noir
    existe = sum(img(yh:yb, c)<= seuil) >0;  %Existe t-il au moins un pixel noir
    c=c+1;
end
debut=c-1;
%-------------------------------------------------
%Detection de la fin du premier caractère
existe = 1;
while (existe) && (c<longueur)              %Recherche d'une colonne entierement blanche
    existe = sum(img(yh:yb, c)<= seuil) >0;   %Existe t-il au moins un pixel noir
    c=c+1;
end
fin_precedent = fin;
fin=c-1;
```

Ce processus est itératif et s'effectue pour chacun des caractères du document.
Une fois le caractère isolé, avant de le traiter on regarde si

- La distance entre le caractère précédant et le suivant est supérieur à 80% de la largeur du caractère, si c'est le cas alors on peut supposer qu'il y ait un espace.
- La taille su caractère doit être supérieur à 12 pixels sinon on considère qu'il ne s'agit pas d'un caractère mais d'une tache.

```
%Analyse du caractere
        if(c<longueur)
            if (debut-fin_precedent > 0.80*(fin-debut) && fin_precedent ~= 0)
                                                %Y a-t-il un espace avant ce caractère
                disp("Espace");
                fprintf(File, ' ');                 %Ecrit un espace
            end

            %Analyse du caractere
               if (yb-yh >12 && fin-debut >12)      %Si caractere assez grand (pas une tache)
                  Image1 = img(yh:yb,debut:fin);     %Delimite le caratère
                  caractere = Detection_caractere(Image1, BDD, Liste_Caractere);  %Detection du
                  caractere
                disp (caractere);
                fprintf(File, caractere);           %Sauvegarde du caratere
            end
        end
    end
    disp('saut de ligne');
    fprintf(File, '\n');                            %Ecrit un saut a la ligne
end
```

IV. Détection du caractère

Une fois le caractère isolé nous appelons la fonction **`detection_caractere`**, qui va recadrer le caractère et faire intercorrélation avec tous les caractères de la base de données.

1. Recadrage

Nous avons défini une fonction **`cadre`** qui prend en entrée une image contenant un seul caractère et renvoie en sortie les délimitations du caractère.

```
function [yh,yb,debut,fin] = cadre(tableau)
```

Par exemple en appliquant cette fonction à l'image suivante :

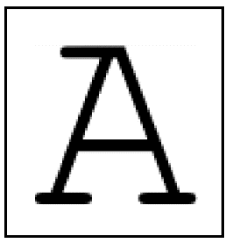

Figure 5 : Image de test pour le cadrage d'une lettre

On obtient :

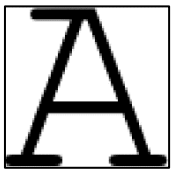

Figure 6 : Rendu de la fonction cadre après application sur la Figure 4

Script

```
function [yh,yb,xg,xd] = cadre(tableau)
    [hauteur,longueur]=size(tableau);
    global seuil;
    %-------------------------------------------------------------------
    %Detection de la ligne haute
    existe = 0;                                         %0 s'il n'y a aucun pixel noir sur la ligne
    l=1;                                                %Commencement à la 1ere ligne
    while (not (existe)) && (l<hauteur)                 %Recherche d'une ligne avec un pixel noir
        existe = sum(tableau(l, :)<= seuil) >0;         %Detection d'un pixel noir sur la ligne l
        l=l+1;
    end
    yh=l-1;
    %-------------------------------------------------------------------
    %Detection de la ligne basse
    existe = 0;
    l = hauteur-1;
    while (not (existe)) && (l>0)                       %Recherche d'une ligne avec un pixel noir
        existe = sum(tableau(l, :)<= seuil) >0;         %Si il y a au moins un pixel noir
        l=l-1;
    end
    yb=l+1;

    %-------------------------------------------------------------------
    %Detection du début du premier caractère
    existe = 0;                                         %0 s'il n'y a aucun pixel noir sur la colonne
    c=1;
    while (not (existe)) && (c<longueur)                %Recherche d'une colonne avec un pixel noir
        existe = sum(tableau(yh:yb, c)<= seuil) >0;
        c=c+1;
    end
    xg=c-1;
    %-------------------------------------------------------------------
    %Detection de la fin du premier caractère
    existe = 0;
    c = longueur;
    while (not (existe)) && (c>0)                       %Recherche d'une colonne avec un pixel noir
        existe = sum(tableau(yh:yb, c)<= seuil) >0;     %Existe t-il au moins un pixel noir
        c=c-1;
    end
    xd=c+1;
end
```

2. Détecter le caractère

Une fois l'image recadrer, l'intercorrélation est réalisée sur chacun des caractères et celui qui a obtenu le score maximum est le caractère le plus probable.

Script

```
function Caractere = Detection_caractere(Image1, BDD, Liste_Caractere)
    %Ajustement de l'image autour du caractere
    [a,b,c,d] = cadre(Image1);
    Image1 = Image1(a:b, c:d);

    %Initialisation
    V_Max =0;
    n=length(Liste_Caractere);

    %Intercorrelation de l'image avec tous les caracteres de la base de donnees
    Image1 = imcomplement(Image1);          %Inversion des couleurs
    Caractere = 'A';                        %Valeur de sécurité si aucun n'est satisfaisant par la suite
    for (i=1:n)
        temp = Intercorrelation2(Image1, BDD{i});        %Intercorrelation
        if temp > V_Max
            V_Max = temp;
            Caractere = Liste_Caractere(i);
        end
    end
end
```

V. Intercorrélation

1. Prérequis

Nous commençons par calculer le rapport de forme des deux images à intercorréler et on suppose que si le rapport de forme est trop différent (supérieur à 15%) alors il ne peut s'agir de caractères similaires, le résultat de l'intercorrélation est par défaut de 0.
Nous avons fait le choix de vérifier ce rapport car faire l'intercorrélation entre deux images suppose d'avoir deux images de même taille donc nous procédons à un redimensionnement de l'image de la base de données or si nous ne vérifions pas le rapport, cette image pourrait être déformée de manière significative et donc conduire à des erreurs (par exemple `o` et `0`). De plus cette solution amène à un gain de temps.

Avant de parvenir à cette solution nous avons essayé de redimensionner l'image sans changer son rapport de forme. Mais cela donnait des erreurs pour deux lettres similaires (par exemple `I` et `T`) car cela manquait d'ajustement. Le résultat était meilleur avec la première solution.

2. Intercorrélation

L'intercorrélation consiste à faire le produit des valeurs pixels par pixels sans oublier de normer le calcul, et on retourne en sortie le score.

Voici la fonction `Intercorrelation2` :

```
%Fonction d'intercorrelation
function Resultat=Intercorrelation2(Image1, Image2)
    [H1, L1] = size(Image1);                          %Calcul des rapports de taille
    [H2, L2] = size(Image2);
    rapport1 = H1/L1;
    rapport2 = H2/L2;
    if (abs(rapport1-rapport2)/rapport1 < 0.15)       %Si les rapport ont une
                                                      difference inferieure a 15%
        Image2 = imresize(Image2, [H1, L1]);
       Resultat           =           sum(sum(Image1.*Image2))/sqrt(sum(sum(Image1.^2
      ))*sum(sum(Image2.^2)));
    else
        Resultat = 0;
    end
    %disp(Resultat)
end
```

VI. Résultats

1. Test sur capture d'écran

On lance alors **OCR3.m** qui est le programme principal, on importe une capture d'écran d'un texte (gauche) contenant les lettres de l'alphabet en majuscule puis en minuscule avec la taille de la police puis on obtient la retranscription (droite) :

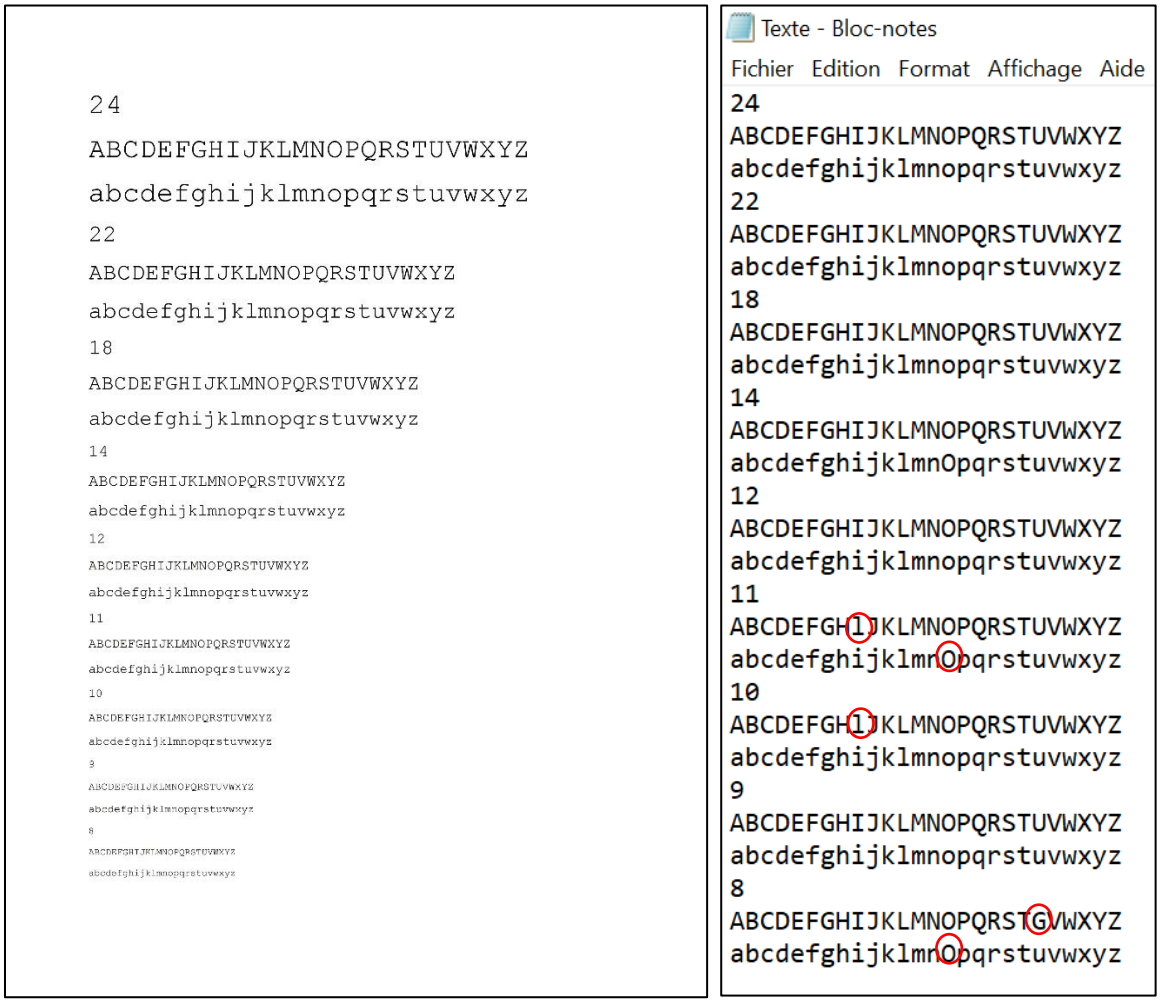

Figure 7 - Image contenant le texte (gauche) / retranscription (droite) pour une capture d'écran

A part le o minuscule qui devient un O majuscule à 2 reprises, le résultat est vraiment satisfaisant.
Erreur : 5/486 soit environ 1%

2. Test sur un scan

Lorsqu'on importe cette fois un scan avec un angle léger contenant une phrase en majuscule puis en minuscule sous différentes tailles :

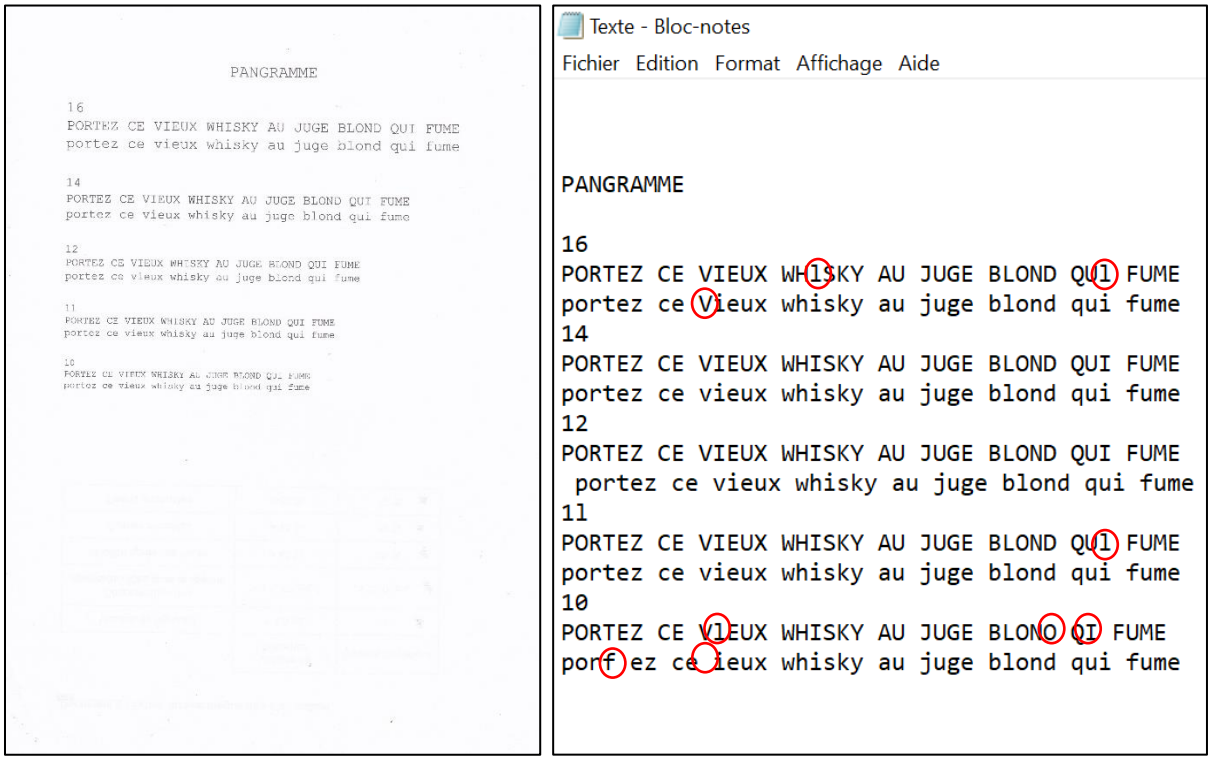

Figure 8 - Image contenant le texte (gauche) / retranscription (droite) dans le cas d'un scan

Il en résulte moins d'erreurs que nous le pensions, en police 10, il manque une lettre minuscule et majuscule, il y a une confusion de lettres (t->f). Pour les autres tailles le I majuscule est parfois retranscrit en l. Autrement la retranscription est prometteuse.
Erreurs : 9/389 soit environ 2,3%

VII. Conclusion

Malgré quelques rares erreurs, notre algorithme de reconnaissance optique de caractères fonctionne très bien pour une capture d'écran. En complexifiant la tâche avec un scan incliné d'un texte imprimé, quelques confusions apparaissent surtout pour une petite taille de police, mais l'algorithme s'en sort toujours bien.
Pour la suite, nous essayerons donc de rajouter les ponctuations et les caractères accentués à notre base de données, nous veillerons à corriger les erreurs de confusions. De plus nous essayerons d'implémenter un vrai traitement des tâches qui pourraient apparaître sur le scan.