Reconnaissance de l'écriture manuscrite
Rapport de projet

Laurent Antoinette, Romain Campillo, Tony Nguyen
L3 informatique
Faculté des Sciences
Université de Montpellier.

12 mai 2023

Table des matières

Table des figures

List of Algorithms

1 Présentation du Sujet

1.1 La problématique

La reconnaissance de l'écriture manuscrite consiste à traduire un texte manuscrit en un texte numérique, interprétable par l'ordinateur. Cette application commence à être utilisée dans différents secteurs, mais la variation des styles d'écritures manuscrites d'une personne à l'autre et la mauvaise qualité du texte pose des défis importants pour leurs numérisations.
Les méthodes classiques étudiées auparavant ou présentées dans la littérature, notamment l'amélioration de la qualité d'images (filtres, etc.) ou la reconnaissance des formes complexes, peuvent ne pas répondre à ce type de problématique.
Ainsi, dans ce projet, nous allons explorer un autre axe de recherche basée sur les méthodes de réseaux de neurones qui ont prouvé leur efficacité à reconnaître et classifier les formes les plus complexes.
Cette recherche va nous permettre de mettre en oeuvre nos compétences en traitement d'image et d'explorer de nouvelles méthodes plus avancées.

1.2 Quelques approches de reconnaissance de formes

1.2.1 k-Nearest Neighbor

La méthode du k-Nearest Neighbor se base sur le principe d'appartenance des échantillons à la même classe, qui ont tendance à se regrouper dans l'espace des caractéristiques. Pour classer une nouvelle entrée, on regarde les k points d'entraînement les plus proches. La nouvelle entrée est classée en fonction de la classe majoritaire des voisins.

1.2.2 Template matching

Le template matching consiste à comparer les images à classer avec des images de référence à l'aide d'une fonction de distance euclidienne

L'avantage de cette solution est qu'il n'est pas nécessaire d'entrainer un modèle au préalable. Elle est néanmoins sensible aux rotations et aux déplacements dans l'image.

1.2.3 Les réseaux de neurones

Les réseaux de neurones sont souvent utilisés pour résoudre des problèmes de classification et de prédiction. Ils sont constitués de couches de neurones interconnectées en s'inspirant du fonctionnement du cerveau humain. Cette méthode nécessite une phase d'apprentissage qui peut être longue, coûteuse et qui repose sur la qualité et la quantité des données d'entraînement fournies.
Nous avons choisi cette solution pour procéder à la reconnaissance des caractères dans notre projet.

1.3 Cahier des charges

1.3.1 Objectifs

Notre objectif est de proposer une méthode pour convertir une image de texte manuscrit en un texte numérique. Dans notre étude nous allons considérer les 26 lettres de l'alphabet latin en majuscule et espacées.

1.3.2 Besoins et contraintes

Les besoins

Capturer une image L'utilisateur pourra prendre des photos par notre logiciel à l'aide d'une webcam. Il peut également importer des images depuis son système de fichier, et ce, à travers une interface Homme-Machine.

Pré-traitement Le logiciel devra améliorer la qualité de l'image capturée en appliquant une réduction bruit et en utilisant des filtres prédéfinis.

Segmentation de l'image Les différents caractères présents sur l'image seront localisés à l'aide de la projection verticale et horizontale de l'histogramme.

Extraction des caractères Chaque caractère présent dans l'image d'entrée sera extrait sous forme d'imagette à l'aide des coordonnées calculées dans la partie segmentation.

Reconnaissance Les caractères, étant sous formes d'imagettes, feront l'objet d'une reconnaissance de façon individuelle grâce à un réseau neuronal convolutif.

Présenter Une fois les caractères manuscrits reconnus en caractères numériques. Ils seront assemblés, puis affichés à l'utilisateur.

Les contraintes Nous nous fixons comme contraintes de ne pas utiliser de services comme Google Collab car Google possède un modèle économique type "Freemium" (initialement gratuit mais avec des fonctinalités payantes). Nous souhaitons créer un logiciel suffisament performant pour qu'il puisse être lancé sur l'une de nos machines personelles.
Une webcam ou un appareil photo numérique est nécessaire.

1.3.3 Résulats attendus

Notre programme doit reconnaître des lettres manuscrites sur un fond blanc. Les lettres seront des caractères majuscules non-liés et sans accent ni caractère spécial.

2 Technologies utilisées

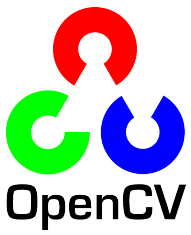
2.1 Langages et outils

2.1.1 Python3

Python3 est un langage de programmation interprété, multiparadigme et multiplatforme. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions. (`https://fr.wikipedia.org/wiki/Python_(langage)`) De par sa simplicité et sa popularité, de nombreuses bibliothèques implémentent ce dont nous avons besoin. Python étant conçu pour programmer rapidement et efficacement, nous pourrons nous concentrer davantage sur la problématique donnée.

2.1.2 OpenCV

OpenCV (Open Computer Vision) est une bibliothèque open source spécialisée dans le traitement d'image en temps réel. Elle est déposée sous licence libre, et sa popularité ainsi que sa simplicité d'utilisation en font un outil adéquat pour notre projet. Parmi ses nombreuses fonctionnalités, nous utiliserons surtout ses fonctions de filtrage et de seuillage.



2.1.3 You Only Look Once

You Only Look Once (YOLO) est une archicture de Réseau Neuronal Convolutif (CNN) qui est capable de localiser des objets dans une image et, en même temps, de les classer.

Les réseaux de neurones Les réseaux de neurone simulent plus ou moins le fonctionnement du cerveau humain. Ils sont constitués d'une multitude de neurones interconnectés entre eux qui reçoivent et renvoient des informations.

Un neurone prend plusieurs entrées pondérées, les somme, puis les passe à travers une fonction d'activation pour produire une sortie vers un autre neurone.

L'architecture des réseaux de neurone simple est organisée en couche. La 1ère couche représente l'entrée et la dernière couche la sortie. Entre le début et la fin du réseau, les couches intermédiaires sont connectées vers l'avant et de façon complète. Un neurone est connecté à tous les neurones de la couche précédente et de la suivante mais pas à ceux de la même couche. L'apprentissage d'un réseau de neurones en couches se fait par rétropropagation, où l'erreur entre la sortie du réseau et la sortie attendue est propagée en arrière dans le réseau, et les poids des connexions entre les neurones sont ajustés pour minimiser cette erreur. Ce processus est répété pour chaque exemple d'entraînement jusqu'à ce que le réseau atteigne un niveau de précision satisfaisant.

Les Réseaux de Neurones Convolutifs (CNN) Contrairement aux réseaux de neurones traditionnels, qui utilisent des connexions denses entre les neurones des couches adjacentes, les CNN utilisent des couches de convolution qui appliquent des filtres pour extraire des caractéristiques importantes de l'image. Cette opération permet d'extraire des motifs et des formes à différentes échelles et positions de l'image. Les couches de sortie du CNN sont des couches denses classiques qui combinent les informations extraites par les couches précédentes pour produire une sortie finale, telle que la classe de l'objet présent dans l'image.

YOLO est connu pour sa grande précision et ses bonnes performances dans la détection des objets dans une scène. Nous n'utiliserons cependant que les fonctionnalités de classification de YOLO puisque nous localisons déjà les lettres dans notre image.

3 Développements Logiciel : Conception, Modélisation, Implémentation

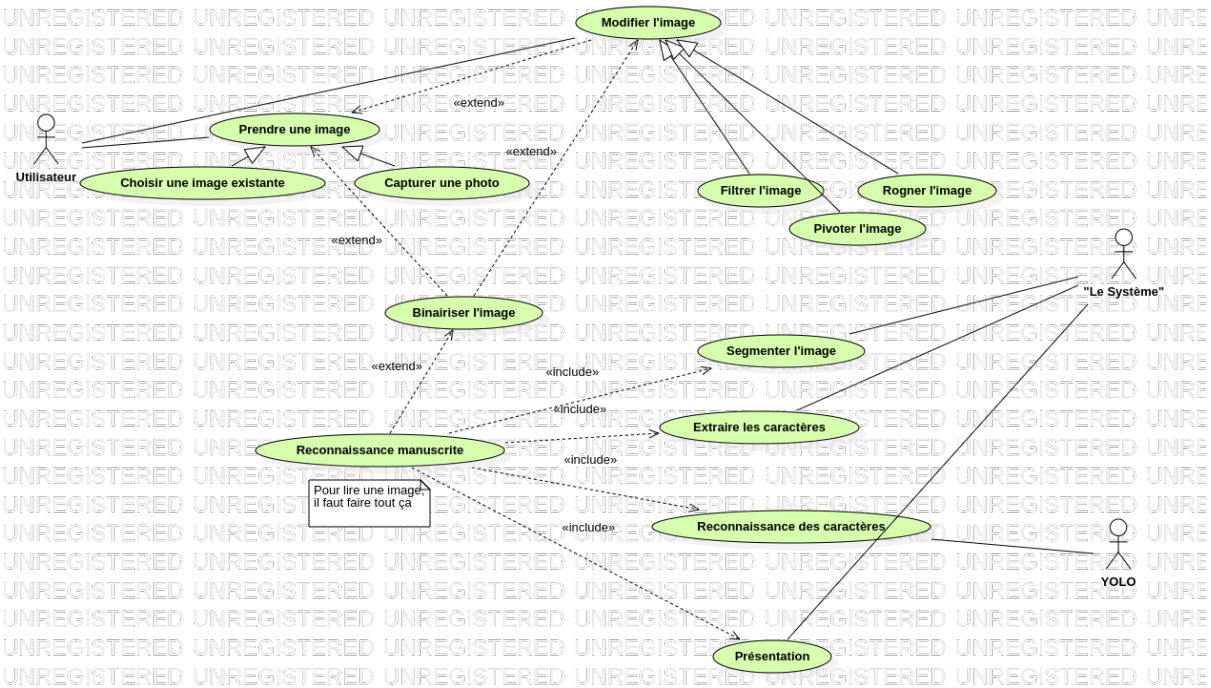


FIGURE 1 – Diagramme de cas d'utilisation

3.1 Interface Homme-Machine

L'IHM nous permet de prendre des photos avec une webcam connectée à l'ordinateur ou de choisir, à partir d'une fenêtre, une image déjà existante.

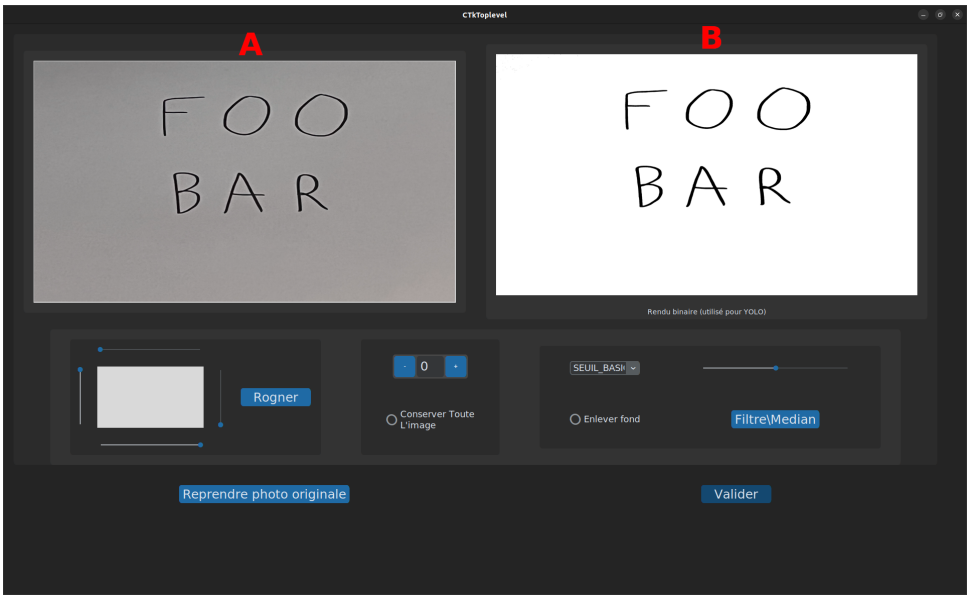


FIGURE 2 – Fenêtre de l'IHM où l'on peut rogner, pivoter et filtrer une image

Après avoir entré une image (figure 2 A), une étape de préprocessing (découpage) peut-être établie pour réduire le bruit concentré sur les bords de l'image et causé par la lumière. L'image résultat (figure 2 B) est ensuite convertie en image binaire réalisée grâce à un seuillage adapté. Puis, les pixels noirs et blancs sont inversés pour faciliter la segmentation des images.

FIGURE 3 – Fenêtre de l'IHM après présentation des caractères reconnus

Après avoir établi le prétraitement, une segmentation des caractères est produite sur une image de bonne qualité (voir figure 3). Chaque caractère est ensuite reconnu séparément par YOLO. L'IHM affiche la lettre la plus probable pour chaque objet détecté.

3.2 Filtres et seuillage

Le pré-traitement de l'image est une phase importante qui affecte le bon déroulement de toutes les étapes futures. Les images prises par les appareils photos souffrent en général de divers problèmes liés à la difficulté de l'acquisition (manque de luminosité, présence de bruit, éclairage inhomogène, texte penché etc.). Puisque chaque image comporte des défauts qui lui sont propres, certains seuils et certains filtres seront plus adaptés que d'autres pour obtenir une binarisation de qualité.

3.2.1 Seuillage global

Le seuillage global consiste à diviser une image (en niveau de gris), en deux parties en utilisant une valeur prédéfinie appelée "seuil global". Chaque pixel prend la valeur 0 (noir) si sa valeur initiale est en dessous du seuil global et 255 (blanc) sinon.

Le seuillage global est une méthode simple qui offre une binarisation sans bruit et qui suffit lorsque l'acquisition est de bonne qualité.

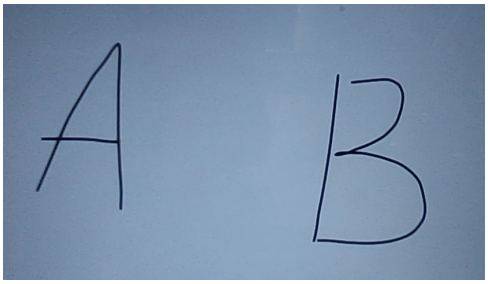


FIGURE 4 – image prise par l'appareil photo

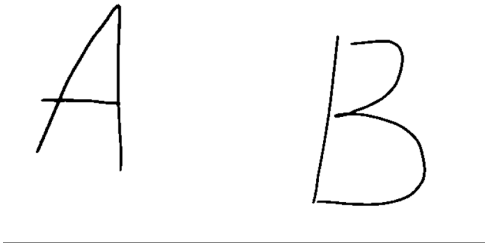


FIGURE 5 – image après seuillage global

Cependant si l'image souffre d'un éclairage inhomogène le seuillage global devient inefficace.

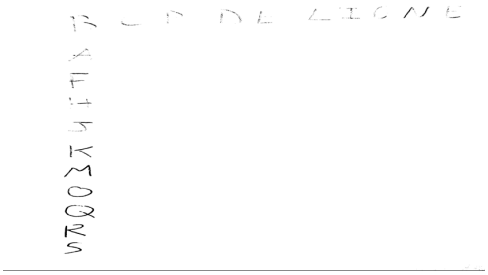


FIGURE 6 – Seuillage global à 80

FIGURE 7 – Seuillage global à 140

3.2.2 Seuillage adaptatif

Contrairement à la méthode de seuillage global, qui utilise un seuil unique pour diviser une image en deux parties, la méthode de seuillage adaptatif utilise un seuil différent pour chaque pixel de l'image en fonction de la valeur de ses voisins. Cela est particulièrement efficace pour exposer les lignes et les contours de l'image, et donc pour exhiber les lettres manuscrites. Il faut noter cependant que la binarisation produite contient du bruit "poivre et sel" (des valeurs anormalement basses ou hautes à certains endroits). Pour pallier ce problème, nous appliquons au préalable un filtre médian sur l'image en niveau de gris.

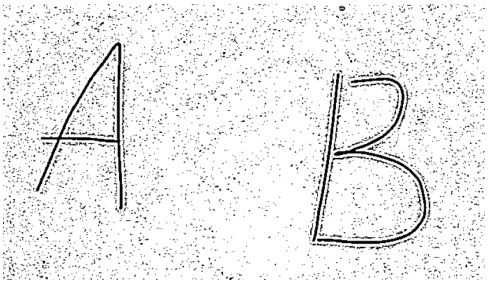


FIGURE 8 – Seuil adaptatif

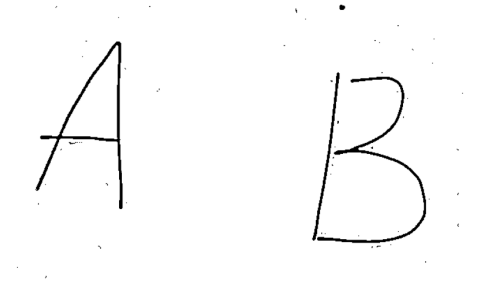


FIGURE 9 – Seuil adaptatif avec un filtre médian

3.2.3 Seuillage de Sauvola

La méthode de Sauvola, qui se base également sur un seuillage local, vise à améliorer la qualité de la binarisation en présence d'un éclairage inhomogène et de variations locales de la luminance. Pour chaque pixel, le seuil est déterminé par cette formule mathématique :

$$T(x,y) = m(x,y) * (1 + k * (s(x,y) / R - 1))$$

où T(x,y) est le seuil pour le pixel à la position (x,y), m(x,y) est la moyenne locale des niveaux de gris, s(x,y) est l'écart type local des niveaux de gris, k est un paramètre de pondération et R est la valeur de l'échelle fixe. Cette méthode est particulièrement adaptée pour la segmentation de l'écriture manuscrite à condition de choisir une bonne valeur pour k.

FIGURE 10 – Seuillage de Sauvola

3.2.4 Seuillage Otsu

La méthode d'Otsu effectue simplement un seuillage global, mais en minimisant l'écart entre le nombre de pixels blancs et le nombre de pixels noirs. Pour la reconnaissance de l'écriture manuscrite, il peut être utile pour différencier le support d'écriture (en général une feuille blanche) du reste de la scène. Pour exploiter au mieux ce seuil, il faut d'abord appliquer un flou gaussien qui égalise la couleur du support d'écriture (supposée uniforme). Il se peut que le seuillage identifie les lettres comme élément du second plan. Dans ce cas, nous effectuons une fermeture (une dilatation suivie d'une érosion) sur l'image seuillée.

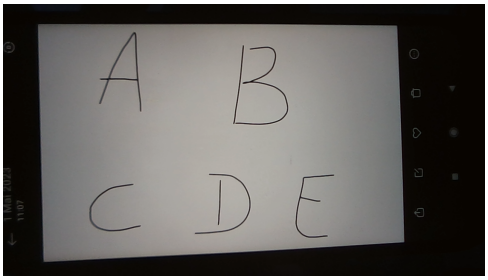


FIGURE 11 – image initiale

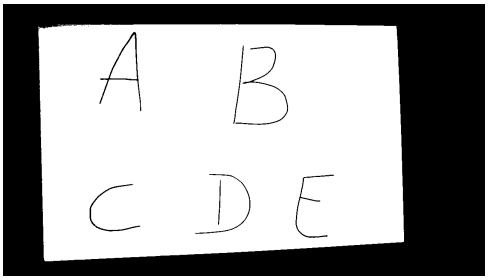


FIGURE 12 – après le seuillage d'otsu

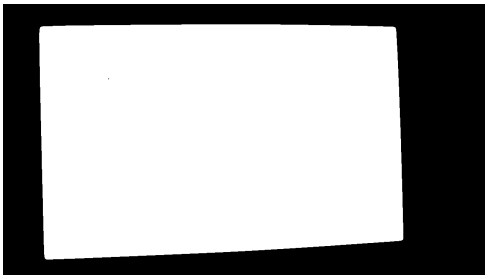


FIGURE 13 – après la fermeture

Après avoir obtenu la position de la feuille, nous appliquons un NON binaire pour nous en servir comme masque. Après un OU binaire entre ce masque et l'image d'origine, on conserve l'information du support d'écriture, et non l'arrière plan.

FIGURE 14 – NON binaire

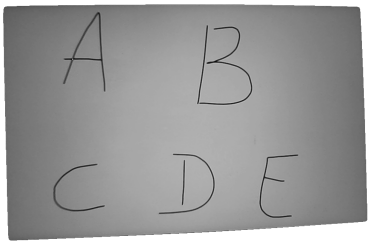


FIGURE 15 – image sans l'arrière plan

3.3 Segmentation

Une fois l'image binarisée, il faut séparer chaque caractère présent dans l'image.

Dans un premier temps, une projection horizontale de histogramme de l'image binaire est calculée afin de segmenter les lignes de texte présentes dessus ; cette projection de l'histogramme correspond à un vecteur de taille égale à la largeur de l'image et sommant le nombre de pixels noirs sur une ligne.
Les images d'entrée ne sont pas toujours parfaites, ce qui provoque l'apparition d'anomalies dans la projection, pouvant ainsi fausser l'analyse de l'image. Face à ce problème, nous avons créé des fonctions permettant de reconnaître et de supprimer ces anomalies que ce soit pour la segmentation des lignes ou des caractères.
Après la segmentation des lignes et en se basant sur l'analyse de la projection verticale de l'histogramme de celles-ci, nous avons implémenté des fonctions permettant la localisation et la segmentation des caractères présents dans les lignes. Le critère de détection est basé sur la succession de projections nulles et non nulles. En sortie, nous optenons l'image de chaque caractère et de chaque ligne. (voir l'algorithme de segmentation 1)

3.4 Entrainement de YOLO

Pour prédire et classer au mieux les caractères segmentés produits, les réseaux de neurones ont besoin d'une première phase d'entrainement à partir d'un grand nombre de données. Concernant les images d'entrainement, nous avons choisi la base de données d'images d'écriture manuscrite développée par le National Institute of Standards and Technology (NIST). Cette base de données contient des images de caractères manuscrits et de formulaires remplis à la main, qui ont été collectés pour soutenir la recherche en reconnaissance d'écriture manuscrite.

Cette banque d'images contient en tout 800 000 photos de caractères manuscrits, écrits par 3600 personnes différentes. Avant d'entrainer notre réseau de neurones, il faut conformer ces images aux images produites par notre segmentation. Nous avons donc enlevé les bords blancs de chaque image de la banque d'images et redimensionné en 128x128.

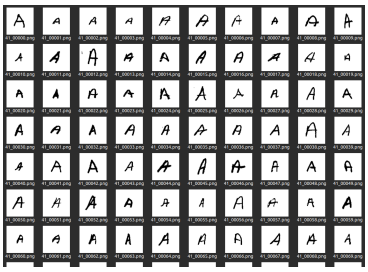


FIGURE 16 – Un exemple pour la lettre A

FIGURE 17 – lettre A après découpage et redimension

Une fois le set de données préparé, nous pouvons passer à l'apprentissage. Nous pouvons paramétrer l'entraînement dans YOLO à l'aide des "hyperparamètres", qui ne sont pas des paramètres appris à partir des données, mais plutôt définis par l'utilisateur avant l'entraînement du réseau. Ils contrôlent divers aspects de l'apprentissage et peuvent affecter considérablement les performances du réseau. Il est important de modifier et de tester des paramètres différents lorsque les résultats ne sont pas suffisament acceptables. Etant donné que nos images sont assez petites et binarisées et que nos objets sont assez faciles à reconnaitre, les paramètres par défaut de YOLO nous suffisent à avoir des résultats convenables.

4 Algorithmes et Analyse

4.1 Segmentation par histogramme de projection

Algorithm 1: SEGMENTATION(image)

Data: image en RGB

Stratégie: la segmentation se produit en deux temps. En premier, la segmentation des lignes avec la projection verticale, et par la suite la segmentation des caractères par la projection horizontale.

Result:

Tableau de couples d'entiers représentant les coordonnées en y de début et de fin des n des lignes sur l'image.

Tableau de tableau de couples d'entiers représentant les coordonnées en x de début et de fin des n lettres contenues dans chaque ligne.

Tableau d'image des lignes

Tableau de tableau d'image des caractères

$ndgImage \leftarrow convertirImageEnNDG(image)$;

/* on met l'image en noir et blanc sauf que le seuillage est inversé */

$imageBinaire \leftarrow binarisationInvers(ndgImage)$;

/* on fait la somme des pixels sur les colonnes et on les divise par 255 */

$projectionHorizontale = (sommeValPixel(imageBinaire, axe = 1))/255$;

$delimitationDesLignes = coordonneeLigne(projectionHorizontal)$;

$imagesLignes = []$;

/* on copie une zone de l'image correspondant à une ligne grâce aux coordonnées calculées */

/* et on l'ajoute à notre tableau d'image de lignes */

for *(x,y) dans delimitationDesLignes* **do**

  imagesLignes.ajouter(ndgImage[x :y, 0 :longeurImage])

end

/* on initialise le tableau de tableaux des images de Caractère on fonction du nombre de ligne detecté */

$imagesCaracteres = [[]pouriallantde0longeur(imagesLignes)]$;

/* on initialise le tableau de tableaux de couple de coordonnées on fonction du nombre de ligne */

$coordonneesCaracteres = [[]pouriallantde0longeur(imagesLignes)]$;

for *index, uneLigne dans imagesLignes* **do**

  $ligneBinaire = binarisationInvers(uneLigne)$;

  /* on fait la somme des pixels sur les lignes de l'image et on les divise par 255 */

  $projectionVerticale = (sommeValPixel(imageBinaire, axe = 0))/255$;

  $delimitationDesCaracteres = coordonneeCaractere(projectionVerticale)$;

  /* on ajoute les coordonnées au tableau de coordonnées */

  $coordonneeCaractere[index].ajouter(delimitationDesCaracteres)$;

  /* on copie une zone de l'image correspondant à un caractère grâce aux coordonnées calculées */

  /* et on l'ajoute à notre tableau d'image de caractère */

  for *(x,y) dans delimitationDesCaracteres* **do**

    $imagesCaracteres[index].ajouter(ndgImage[0 : hauteurUneLigne, x : y])$;

  end

  $redimensionnerImage(binarisation(imagesCaracteres[index], hauteur = 128, longeur = 128), seuil = 127)$;

end

return *delimitationDesLignes, delimitationDesCaracteres, imagesLignes, imagesCaracteres*

FIGURE 18 – Exemple d'une image après filtrage

FIGURE 19 – Images de nos deux lignes créées à partir de l'image de départ

Sur la figure 18, nous voyons une image de bonne qualité prête à être segmentée par l'algorithme n°1 (page13).

Grâce à une projection verticale, nous obtenons ce tableau (figure 20). La 1ère ligne de ce tableau contient autant de tuples que le nombre de lignes. Chaque tuple contient la composante en ordonnée du début et de la fin d'une ligne. Ensuite, on extrait nos 2 lignes comme sur la figure 19 depuis l'image d'entrée.

```
          0.,  0.])
coord_ligne=[(96, 224), (330, 454)]
coord_caracteres=[(337, 465), (522, 664), (739, 900)]
coord_caracteres=[(368, 451), (522, 647), (744, 840)]
```

FIGURE 20 – Structure de données dans l'algorithme SEGMENTATION

Après avoir isolé les lignes, on procède à la séparation des caractères contenus dans chaque lignes en suivant l'algorithme n°2 page 15

Étant donné que l'image est binarisée, lorsqu'un caractère est présent à un endroit donné, cela signifie que la projection verticale de l'histogramme est supérieure à 0 à cette position. L'algorithme n°2 va parcourir cette projection de l'histogramme, déctecter les pics et récupérer l'indice de début et de fin de ces derniers qui correspondront à la limite gauche et droit d'un caractère sur l'image. Nous pouvons voir le résultat de cette fonction sur la figure 20 et des histogrammes de projection horizontale dans l'annexe (page 20).

Algorithm 2: COORDONNEECARACTERE(T)

Data: tableau de flottant représentant la projection vertical d'une image
Stratégie: Trouver les zones non nul dans l'histogramme
Result: tableau de couple d'entier indiquant les coordonnées du début et de la fin d'une lettre

```
coordCaractere = [];
dansLeCaractere = False;
/* Début = pair Fin = impair                                                    */
DF = [];
for i de 0 à taille(T) do
    if T[i] ≠ 0 then
        if !dansLeCaractere or (dansLeCaractere and i == taille(T)-1) then
            DF.ajouter(i);
            if i == taille(myprojection)-1 et taille(DF) mod 2 != 0 then
                supprimer(DF[taille(DF) − 1]);
            end
        end
        if i == len(myprojection)-1 and len(coordDF)coordDF.pop(-1)
            dansLeCaractere = True;
    end
    else
        if dansLeCaractere then
            DF.ajouter(i)
        end
        dansLeCaractere = False;
    end
end
lettres ← [ (DF[i],DF[i+1]) pour i de 0 à taille(DF) avec un pas de 2 ];
if taille(lettres)>1 then
    tailleEspacesEntrelettres = [lettres[i + 1][0] − lettres[i][1]pouride0taille(lettres) − 1];
    poidsEspace = trillageCroissant(tailleEspacesEntrelettres);
    SommeIndice ← 0;
    for i de 0 à taille(poidsEspace)-1 do
        poidsEspace[i] ← poidsEspace[i] ∗ (taille(poidsEspace) − i);
        SI ← SI + i;
    end
    moyenneEspaces ← somme(poidsEspace)/SI;
    j ← 0;
    while tailleEspacesEntrelettres != [] do
        if tailleEspacesEntrelettres < moyenneEspaces ∗ 0.2 then
            D = lettres[j]; supprimer(lettres[j]);
            F = lettres[j]; supprimer(lettres[j]);;
            /* inserer(tableau, indice, élément)                                */
            inserer(lettres, j, (D[0], F[1]));
        end
        else j ← j + 1;
        supprimer(tailleEspacesEntrelettres[0]);
    end
end
return lettres
```

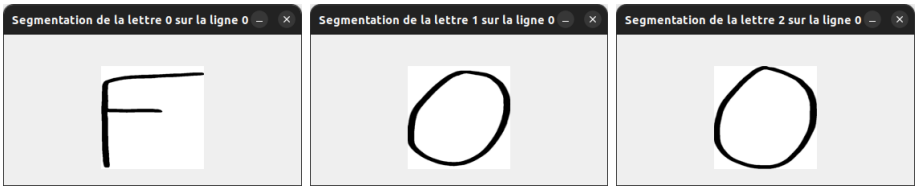


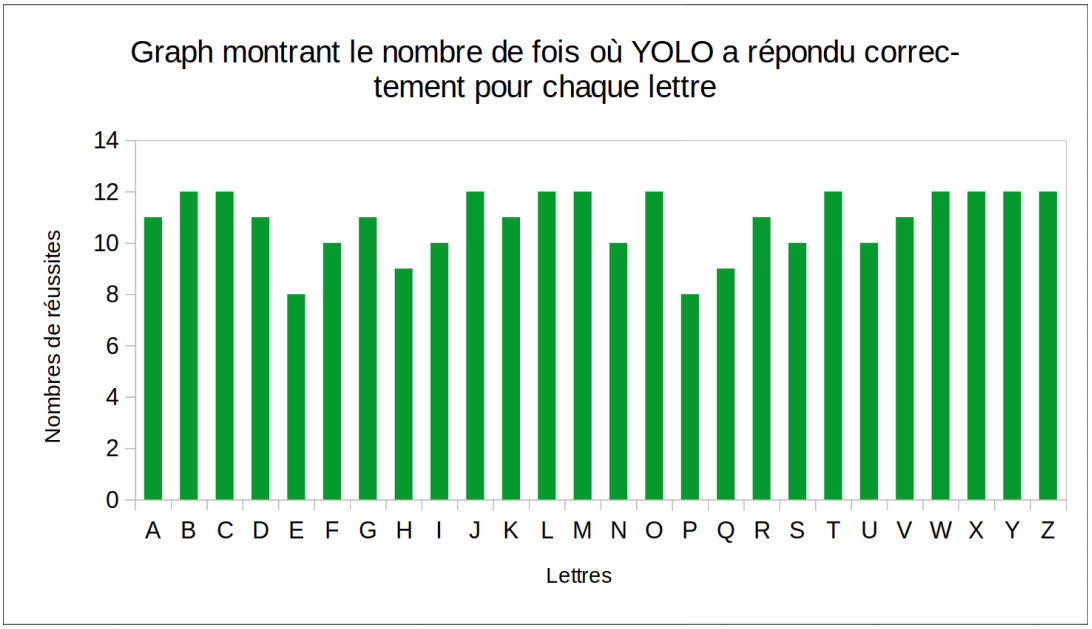
FIGURE 21 – Imagettes des caractères créées à partir de la 1e ligne

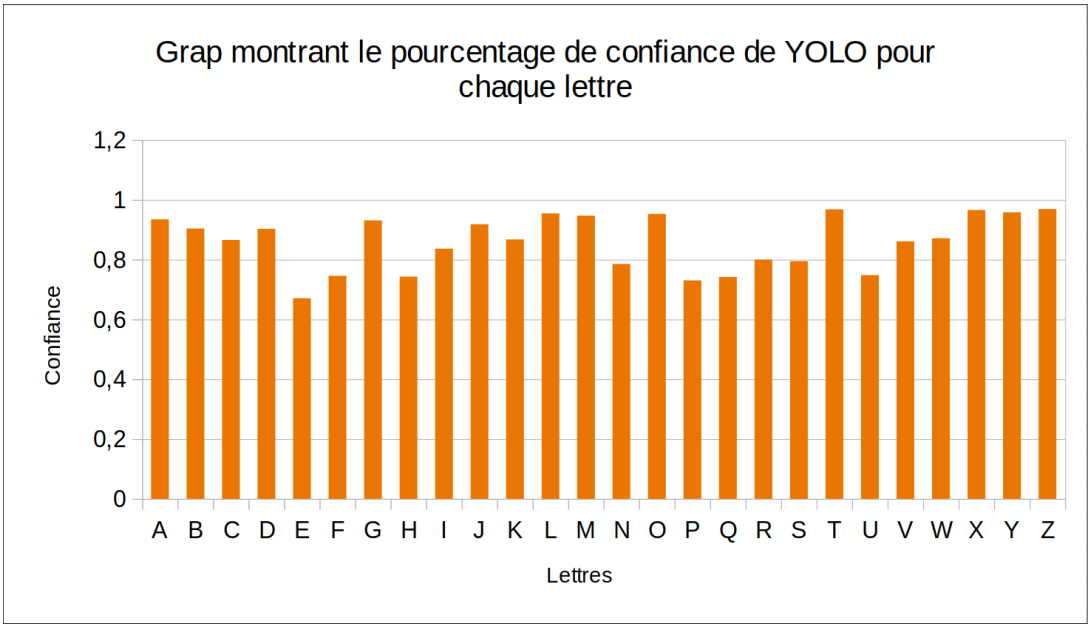
FIGURE 22 – Imagettes des caractères créées à partir de la 2e ligne

5 Analyse des Résulats

5.1 Mesure expérimental de YOLO

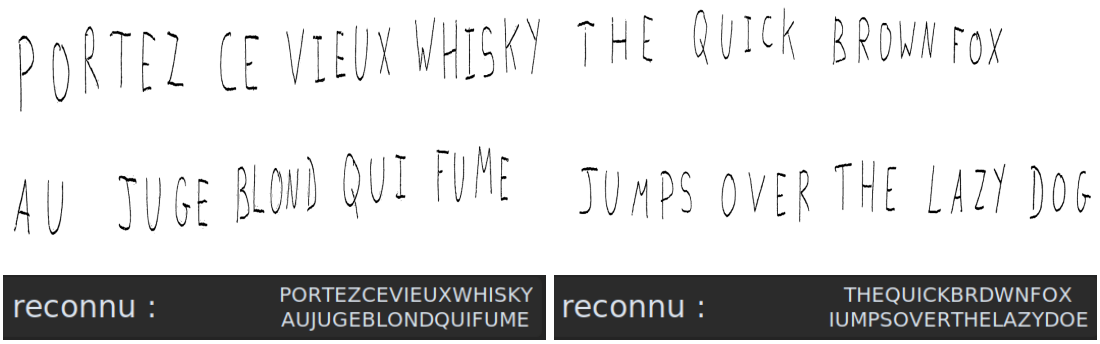
Pour mesurer le taux de prédiction correcte de YOLO, nous avons écrit des lettres de différentes manières et nous avons observé le taux de réussite.





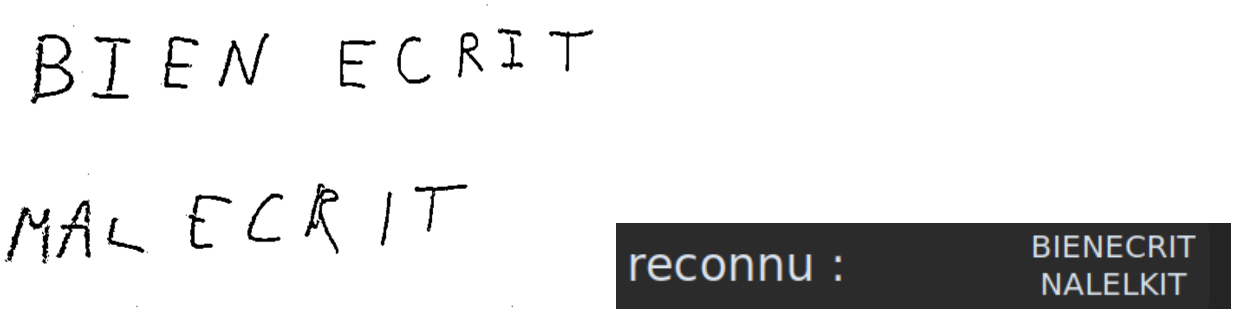
YOLO est en général très confiant et arrive dans la plupart des cas à reconnaitre la bonne lettre (dans de bonnes conditions).

Nous avons utilisé des pangrammes (phrases contenant toutes les lettres de l'alphabet) pour tester notre logiciel.



Pangrammes français et anglais

En pratique toutes les lettres sont bien reconnues. En revanche, des caractères mal écrits ou une image issue d'une binarisation médiocre rendent difficile la reconnaissance par YOLO.



Grâce aux bonnes performances de YOLO, la reconnaissance d'un caractère est de l'ordre de la milliseconde. La reconnaissance se fait donc instantanément.

```
Speed: 0.3ms preprocess, 4.9ms inference, 0.0ms postprocess per image at shape (1, 3, 128, 128)
```

FIGURE 23 –

Bilan et Conclusion

Dans le cadre du projet, nous avons créé une interface graphique qui permet de faire l'acquisition d'une image. Notre logiciel propose quelques méthodes de seuillage adaptées et des outils pour découper et pivoter l'image afin de faciliter la segmentation. Enfin, notre réseau de neurones devine chaque caractère avec une précision convenable et ce, presque immédiatement.

Nous avons respecté le cahier des charges en réalisant un logiciel qui fonctionne et qui produit une réponse convenable au problème posé. Cependant, une contribution de l'utilisateur est souvent nécessaire pour choisir les bons filtres, appliquer une rotation ou encore découper l'image. En outre, notre logiciel reconnait seulement 26 lettres majuscules et ignore les espaces. Reconnaitre les caractères minuscules ainsi que les chiffres et certains caractères spéciaux ne serait pas une tâche difficile à condition d'avoir une grande banque de données et assez de puissance de calcul pour la phase d'apprentissage. Nous pourrions également explorer des approches de post-traitement plus sophistiquées pour améliorer la précision de la reconnaissance des caractères, telles que la correction d'erreurs par les dictionnaires ou l'utilisation de modèles de langue pour l'analyse contextuelle du texte. En revanche reconnaitre les lettres dans une écriture cursive est une tâche bien plus complexe qui reste un domaine de recherche actif en reconnaissance de l'écriture manuscrite.

Annexes

FIGURE 24 – Histogramme de projection vertical

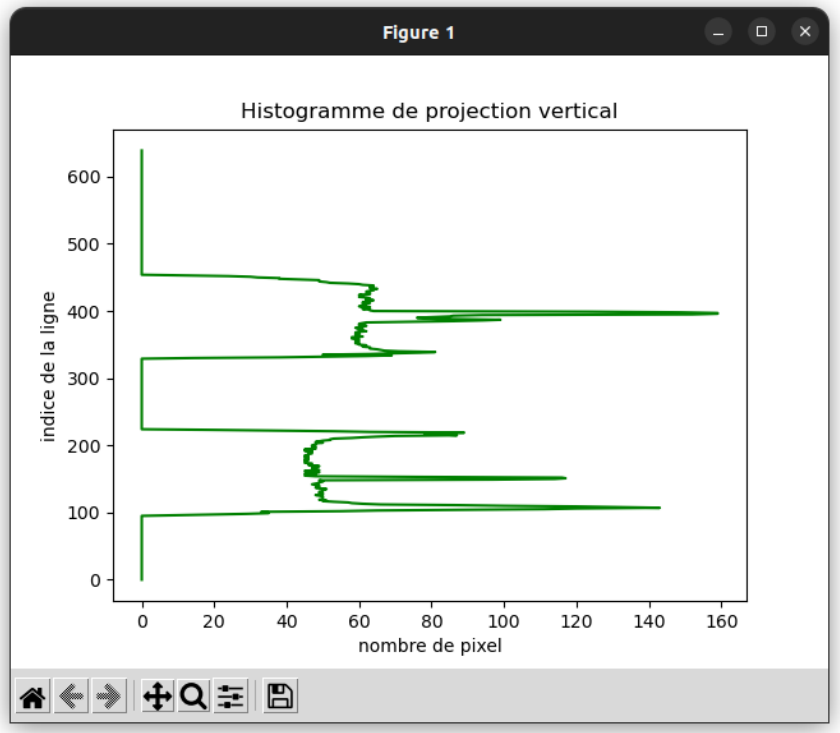


FIGURE 25 – Histogramme de projection horizontal

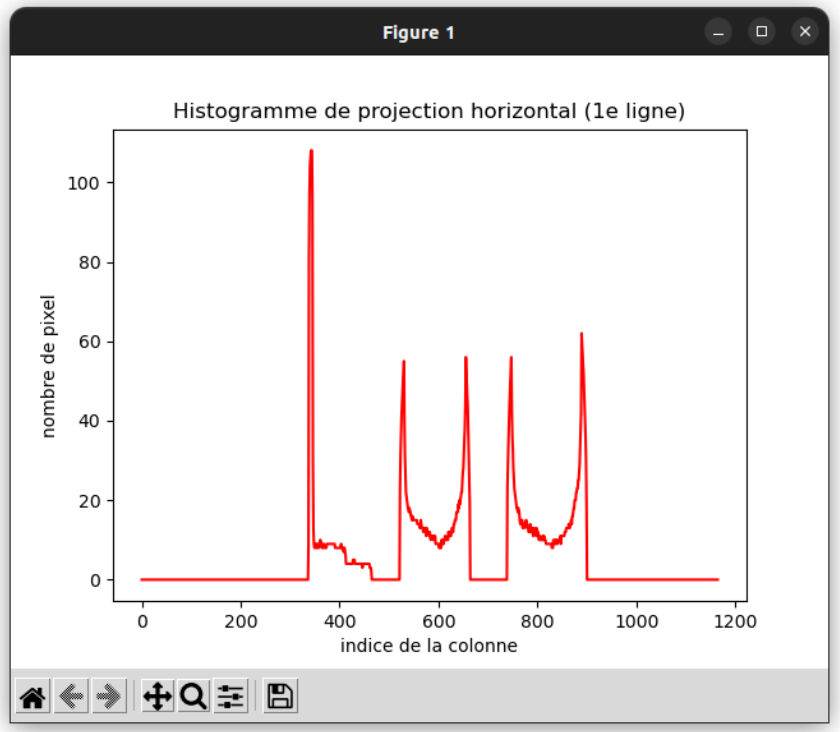


FIGURE 26 – Histogramme de projection horizontal