

ruled

Reconnaissance de l'écriture manuscrite

Rapport de projet

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

May 12, 2023



Contents

1	Présentation du Sujet	5
1.1	La problématique	5
1.2	Quelques approches de reconnaissance de formes	5
1.2.1	k-Nearest Neighbor	5
1.2.2	Template matching	5
1.2.3	Les réseaux de neurones	5
1.3	Cahier des charges	5
1.3.1	Objectifs	5
1.3.2	Besoins et contraintes	6
1.3.3	Résultats attendus	6
2	Technologies utilisées	6
2.1	Langages et outils	6
2.1.1	Python3	6
2.1.2	OpenCV	6
2.1.3	You Only Look Once	7
3	Développements Logiciel : Conception, Modélisation, Implémentation	8
3.1	Interface Homme-Machine	8
3.2	Filtres et seuillage	9
3.2.1	Seuillage global	9
3.2.2	Seuillage adaptatif	10
3.2.3	Seuillage de Sauvola	10
3.2.4	Seuillage Otsu	11
3.3	Segmentation	12
3.4	Statistiques	12
3.5	Entraînement de YOLO	12
4	Algorithmes et Analyse	12
4.1	Segmentation par histogramme de projection	12
5	Analyse des Résultats	15
5.1	Mesure expérimental de YOLO	15
6	Gestion du Projet	16
	Bilan et Conclusions	16

List of Figures

1	Diagramme de cas d'utilisation	8
2	Fenêtre de l'IHM où l'on peut rogner, pivoter et filtrer une image	8
3	Fenêtre de l'IHM après présentation des caractères reconnus	9
4	après la fermeture	11
5	Exemple d'une image après filtrage	13
7	Structure de données dans l'algorithme SEGMENTATION	13
6	Images de nos 3 lignes crée à partir de l'image de départ	14
8	Imagettes des caractères crée à partir de la 1e ligne	14
9	Imagettes des caractères crée à partir de la 2e ligne	14
10	15
11	15
12	Histogramme de projection vertical	18
13	Histogramme de projection horizontal	18
14	Histogramme de projection horizontal	19

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. Bien que cette application commence à être utilisée dans différents secteurs, 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 neurone qui ont prouvé leur efficacité à reconnaître et classer 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.

Le traitement automatique de l'information numérique est particulièrement intéressant pour des étudiants en informatique car il peut être abordé de nombreuses manières.

jsp mais faut reformuler

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 que les échantillons appartenant à la même classe 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

Cette approche est l'une des plus simple. Voici en quoi elle consiste :

Nous allons partir d'une banque d'image de référence. À chaque fois que nous souhaitons classer une nouvelle image, nous allons comparer cette nouvelle image avec toute nos images de référence à l'aide d'une simple fonction de distance euclidienne.

L'avantage de cette solution est qu'il n'est pas nécessaire d'entraîner 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 interconnectés 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 reconnaître les lettres 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. Mais il pourra également utiliser des images depuis son système de fichier. Tout cela à travers une interface Homme-Machine.

Pré-traitement Le logiciel réduira le bruit et rendra l'image plus nette à l'aide de plusieurs filtres prédéfinis.

Segmentation de l'image Les différents caractères présents sur l'image seront localisés à l'aide d'histogramme de projection.

Extraction des caractères Les caractères seront ensuite découpés pour former leurs propres imasettes i.e. une image qui contient TEST devriendra 4 petites imasettes contenant respectivement T E S T.

Reconnaissance Les imasettes feront l'objet d'une reconnaissance de façon individuelle. La solution que nous choisissons d'implémenter est un réseau neuronal convolutif.

Présenter Une fois que les imasettes sont reconnues en caractère. Il est nécessaire de les assembler et d'afficher à l'utilisateur les mots reconnus.

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 fonctionnalités payantes). Nous souhaitons créer un logiciel suffisamment performant pour qu'il puisse être lancé sur l'une de nos machines personnelles.

De plus, une webcam est nécessaire, ou alors un appareil photo numérique.

1.3.3 Résultats attendus

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

2 Technologies utilisées

2.1 Langages et outils

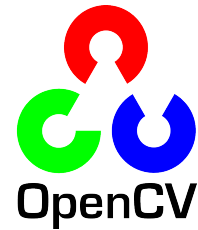
2.1.1 Python3

Python3 est un langage de programmation interprété, multiparadigme et multiplateformes. 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_\(langage\)](https://fr.wikipedia.org/wiki/Python_(langage)))

Nous avons choisi ce langage car il est simple à comprendre et relativement facile à lire. De plus, par sa popularité, de nombreuses bibliothèques implémentent ce dont nous avons besoin. Python étant conçu pour programmer rapidement et efficacement, nous pourrions nous concentrer davantage sur l'implémentation (à redire).

2.1.2 OpenCV

OpenCV (Open Computer Vision) est une bibliothèque open source spécialisée dans le traitement d'images 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 architecture de Réseau Neuronal Convolutif (CNN) qui est capable de localiser des objets dans une image et, en même temps, de les classifier.

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.

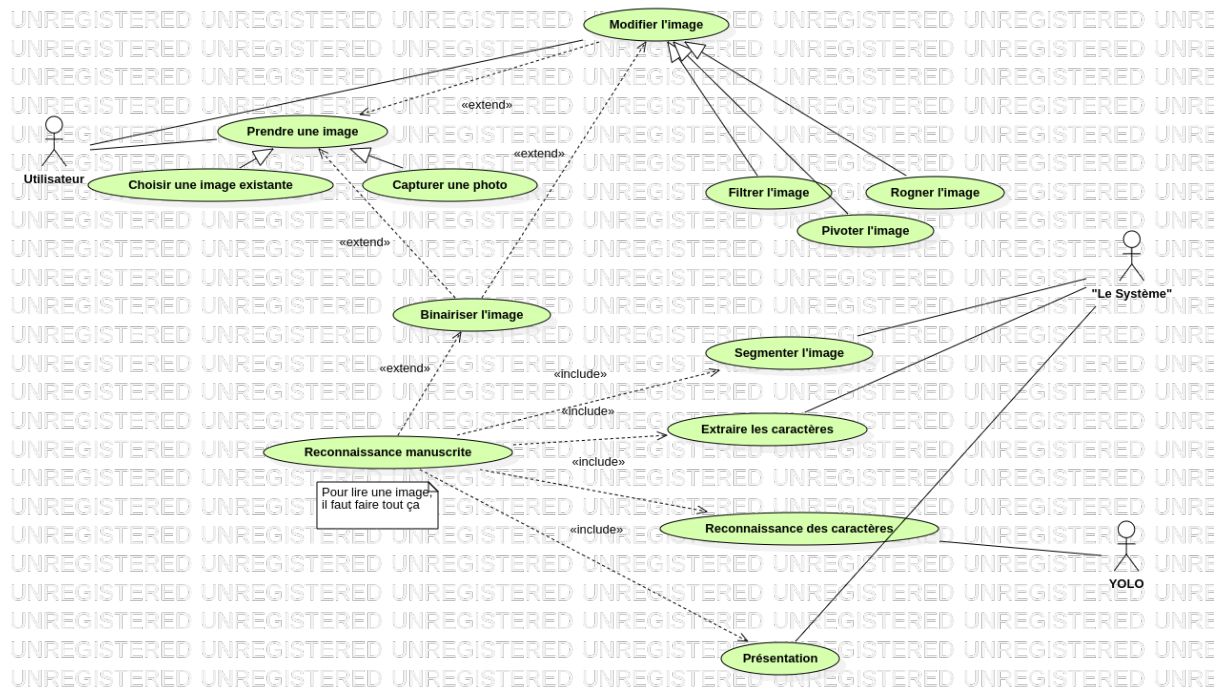
L'architecture des réseaux de neurone simple est organisée en couche. La 1ère couche représente l'entrée et la dernière couche la sortie. Entre le début et la fin du réseau, les couches intermédiaires sont connectées vers l'avant et de façon complète. Un neurone est connecté à tous les neurones de la 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. Etant donné que nous localisons déjà les lettres dans notre image, nous n'utiliserons cependant que les fonctionnalités de classification de YOLO, qui nous suffisent amplement.

3 Développements Logiciel : Conception, Modélisation, Implémentation

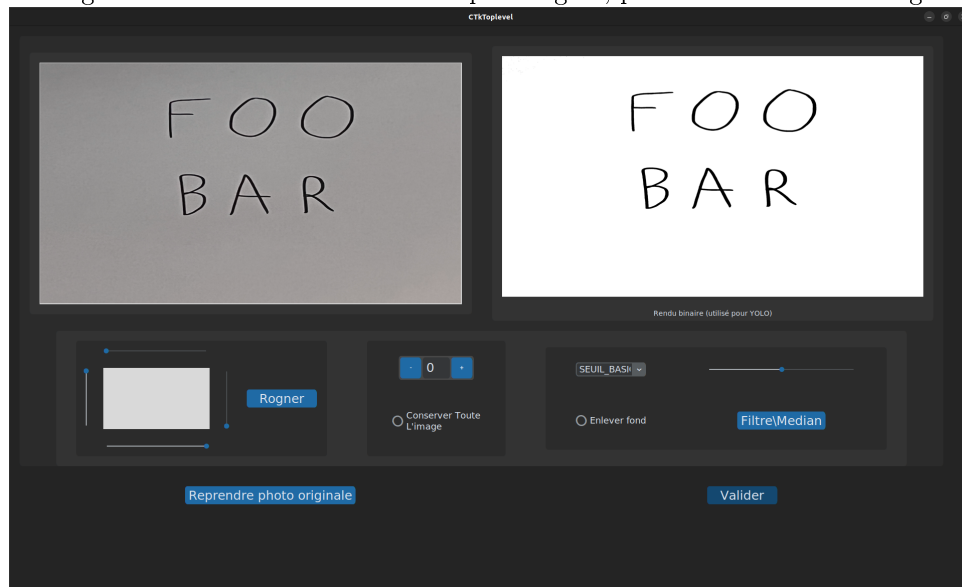
Figure 1: Diagramme de cas d'utilisation



3.1 Interface Homme-Machine

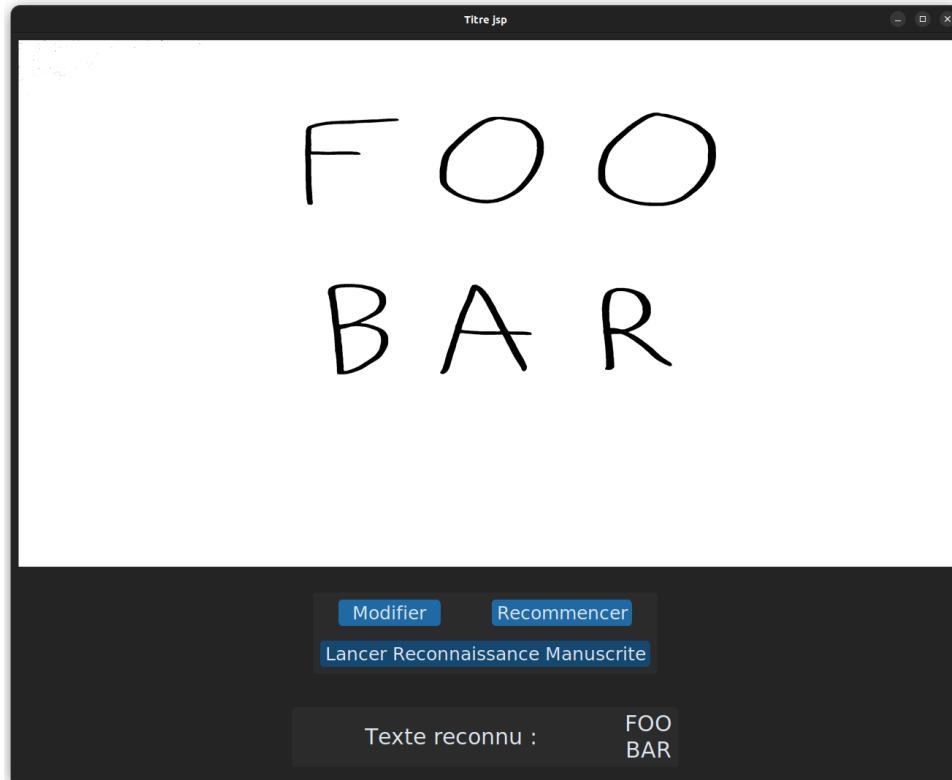
L'IHM nous permet de prendre des photos avec une webcam connectée à l'ordinateur ou choisir à partir d'une fenêtre une image déjà existante.

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 concentrer sur les bords de l'image et causé par la lumière . L'image résultat est convertie après en image binaire réalisé avec un seuil adapté. Puis, les pixels noirs et blancs ont été inversé pour faciliter la segmentation des images.

Figure 3: Fenêtre de l'IHM après présentation des caractères reconnus



Après avoir établi le prétraitement détaillé ci-dessus, une segmentation des caractères est produite. 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. (mettre les images)

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

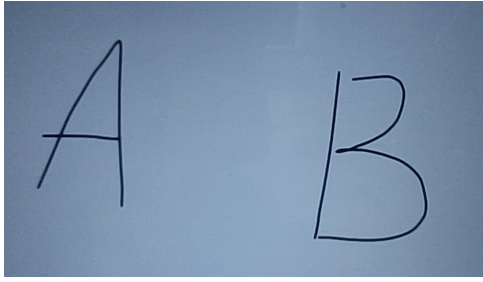


Figure 4: image prise par l'appareil photo



Figure 5: image après seuillage global

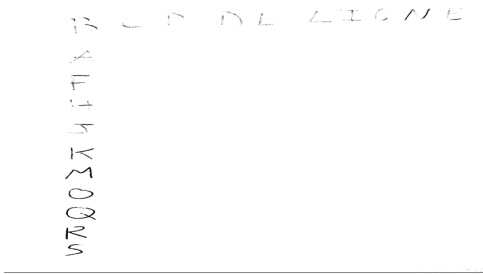


Figure 6: Seuillage global à 80

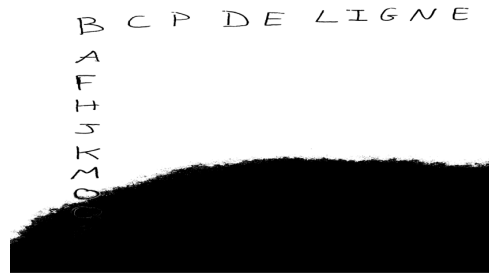


Figure 7: Seuillage global à 140

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 haute à certains endroits). Pour pallier ce problème, nous appliquons au préalable un filtre médian sur l'image en niveau de gris.

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 .

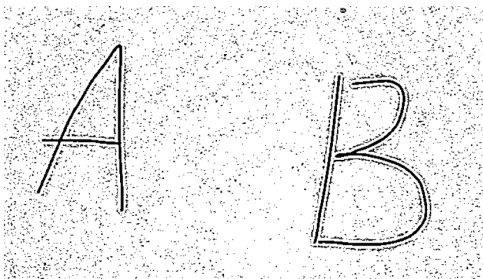


Figure 8: Seuil adaptatif

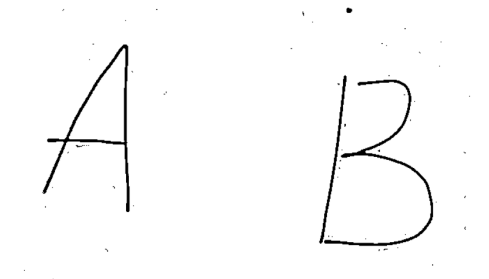


Figure 9: Seuil adaptatif avec un filtre gaussien

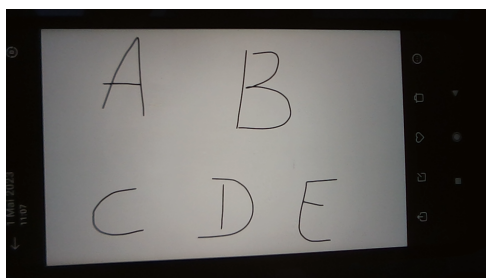


Figure 10: image de base

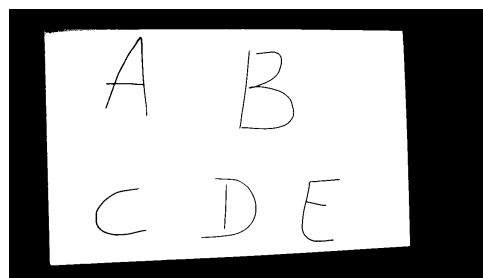
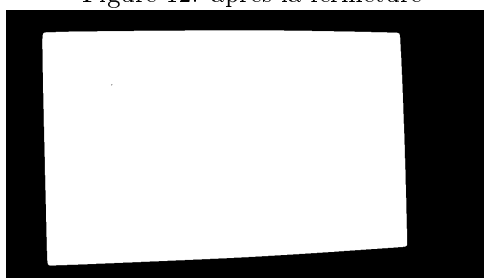


Figure 11: après seuil otsu

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 12: apres 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, le texte apparaît sans la scène (mal dit à reformuler)



Figure 13: NON binaire

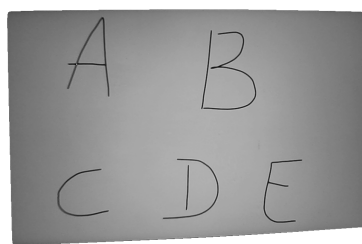


Figure 14: image Sans le fond

3.3 Segmentation

Une fois l'image binarisée, il faut maintenant 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 implémenter 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. (faudrait dire "voir algo en référençant la bonne page)

3.4 Statistiques

- Nombres de lignes de code :
- Nombres de script

3.5 Entraînement de YOLO

TBD

4 Algorithmes et Analyse

4.1 Segmentation par histogramme de projection

Stratégie SEGMENTATION(image) image en RGB 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.

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

Tableau de tableau de couple d'entier 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 $projectionHorizontale = (sommeValPixel(imageBinaire) / 255)$ $delimitationDesLignes = coordonneeLigne(projectionHorizontale)$ $imagesLignes = []$ 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 (x,y) dans $delimitationDesLignes.imagesLignes.ajouter(ndgImage[x:y, 0:longueurImage])$ on initialise le tableau de tableaux des images de Caractère on fonction du nombre de ligne detecté $imagesCaracteres = [[]pouriallantde0longueur(imagesLignes)]$ on initialise le tableau de

tableaux de couple de coordonnées on fonction du nombre de ligne $coordonneesCaracteres = [[]pouriallantde0longueur(imagesLignes)]$ on initialise le tableau de tableaux de couple de coordonnées on fonction du nombre de ligne $coordonneesCaracteres = [[]pouriallantde0longueur(imagesLignes)]$

$(index, uneLigne dans imagesLignes)$ $ligneBinaire = binarisationInvers(uneLigne)$ 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 (x,y) dans $delimitationDesCaracteres.imagesCaracteres[index].ajouter(ligneBinaire[x:y, 0:longueurUneLigne])$ $redimensionnerImage(binairisation(imagesCaracteres[index], hauteur = hauteurUneLigne, x : y))$ $redimensionnerImage(binairisation(imagesCaracteres[index], hauteur = hauteurUneLigne, x : y))$

$redimensionnerImage(binairisation(imagesCaracteres[index], hauteur = hauteurUneLigne, x : y))$ $redimensionnerImage(binairisation(imagesCaracteres[index], hauteur = hauteurUneLigne, x : y))$ $delimitationDesLignes, delimitationDesCaracteres, imagesLignes, imagesCaracteres$

Figure 15: Exemple d'une image après filtrage



StratStratégie COORDONNEECARACTERE(T) tableau de flottant représentant la projection vertical d'une image Trouver les zones non nul dans l'histogramme 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* = [] *i* de 0 à *taille(T)* *T[i] ≠ 0* !*dansLeCaractere* **or** (*dansLeCaractere* **and** *i* == *taille(T)*-1) *DF.ajouter(i)* *i* == *taille(myprojection)*-1 et *taille(DF)* mod 2 != 0 *supprimer(DF[taille(DF)-1])* **if** *i* == *len(myprojection)*-1 and *len(coordDF)* *coordDF.pop(-1)* *dansLeCaractere* = True *dansLeCaractere* *DF.ajouter(i)* *dansLeCaractere* = False lettres ← [(*DF[i]*,*DF[i+1]*) pour *i* de 0 à *taille(DF)* avec un pas de 2] *taille(lettres)* > 1 *tailleEspacesEntrelettres* = [*lettres[i+1][0]* - *lettres[i][1]* pour *i* de 0 à *taille(lettres)* - 1] *poidsEspace* = *trillageCroissant(tailleEspacesEntrelettres)* *SommeIndice* ← 0 *i* de 0 à *taille(poidsEspace)* - 1 *poidsEspace[i]* ← *poidsEspace[i] * (taille(poidsEspace) - i)* *SI* ← *SI* + *i* *moyenneEspaces* ← *somme(poidsEspace)/SI* *j* ← 0 *tailleEspacesEntrelettres* != [] *tailleEspacesEntrelettres* < *moyenneEspaces* * 0.2 *D* = *lettres[j]*; *supprimer(lettres[j])* *F* = *lettres[j]*; *supprimer(lettres[j])*; *inserer(tableau, indice, élément)* *inserer(lettres, j, (D[0], F[1]))* *j* ← *j* + 1 *supprimer(tailleEspacesEntrelettres[0])* lettres

Sur la figure 5, nous voyons une image de bonne qualité prête à être segmenter par l'algorithme n°4.1 (page 12).

Grâce à une projection vertical, nous obtenons ce tableau (voir figure 7). La 1er ligne de ce tableau contient autant de tuples que le nombre de lignes. Chaque tuples contient la composante en ordonnée du début et de la fin d'une ligne. En choisissant arbitrairement comme largeur d'une ligne, la largeur de l'image, on extrait nos 2 lignes comme sur la figure 6.

Maintenant que nous avons isoler les lignes, nous allons maintenant découper les lettres de chaque lignes en suivant

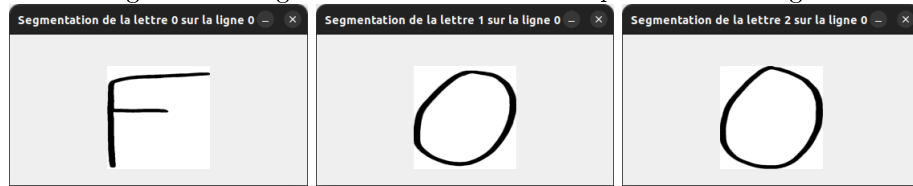
Figure 17: Structure de données dans l'algorithme SEGMENTATION

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

Figure 16: Images de nos 3 lignes crée à partir de l'image de départ



Figure 18: Imagettes des caractères crée à partir de la 1e ligne



l'algorithme n°4.1 page 13

Étant donnée que l'image est filtré, en noir et blanc et binairisé, lorsqu'une lettre est présente à un endroit donnée, cela signifie que sur la projection vertical, l'histogramme est supérieur à 0 à cet endroit là. Sinon, il n'y a pas de lettre et l'histogramme est à 0. L'algorithme n°4.1 va parcourir totalement l'histogramme de projection et va tenté de detecter les pics et récupérer l'indice de début et de fin de ces zones qui correspondront à la limite gauche et droit de la lettre sur l'image. Nous pouvons voir le résultats de cette fonction sur la figure 7 et des histogrammes de projection horizontal dans l'annexe (page 18).

Figure 19: Imagettes des caractères crée à partir de la 2e ligne



5 Analyse des Résultats

5.1 Mesure expérimental de YOLO

Nous procédure de test est la suivante : Écrire des lettres de moins en moins bien, ???, YOLO, graph

Figure 20:

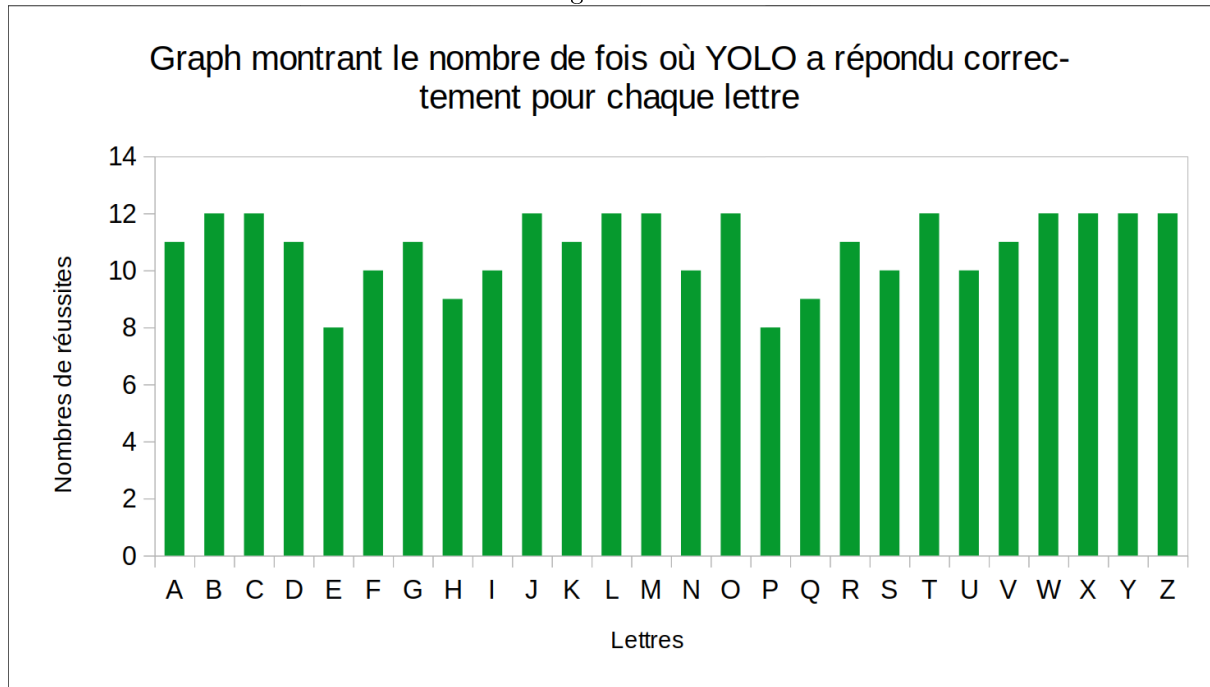
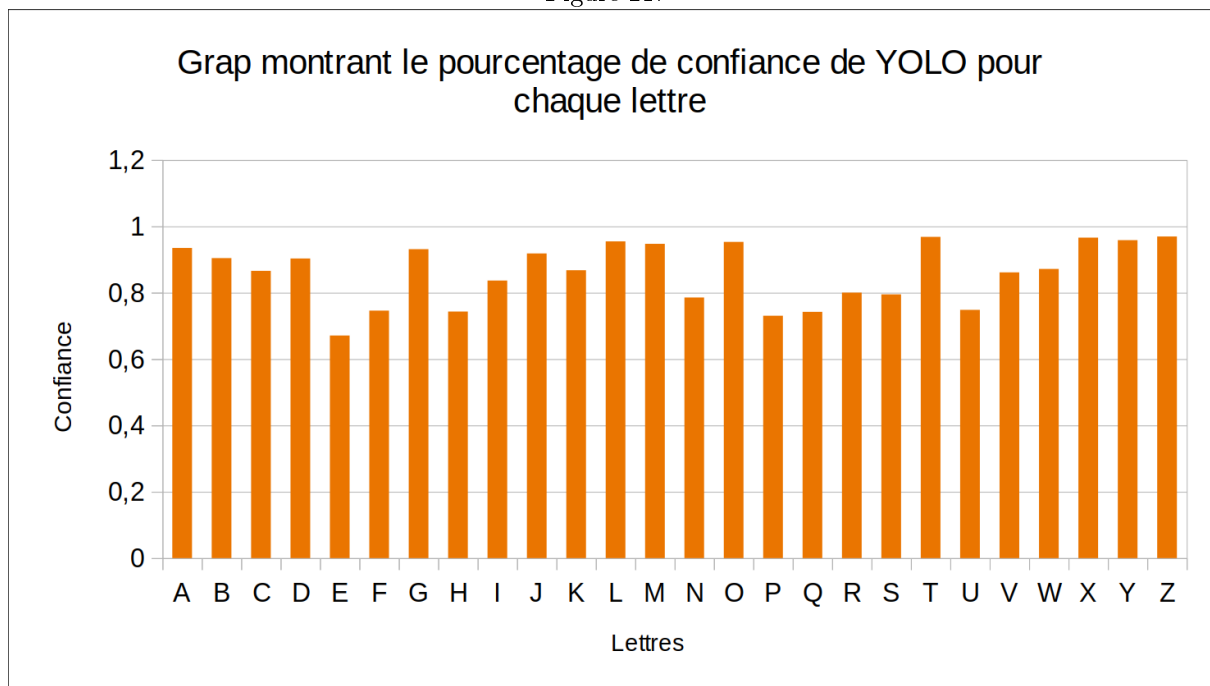


Figure 21:



6 Gestion du Projet

Pendant ce projet, nous avons dû communiquer et collaborer ensemble afin d'atteindre notre but commun. Voici les outils que nous avons utilisés.

Premièrement, pour communiquer nous avons utilisé l'application Discord. Sur lequel nous avons un groupe où nous nous envoyons des messages, des images et des fichiers.

Ensuite, pour le partage et le versioning, nous utilisons git et Github
Laurent : filtrage, segmentation
Romain : IHM, YOLO
Tony : rapport

Bilan et Conclusions

tests

Cependant, certains points laissent à désirer dans notre projet. Malheureusement, une contribution de l'utilisateur est nécessaire. Pour une bonne reconnaissance des caractères, l'utilisateur doit rogner l'image. Il serait plus pratique que le *rognage soit fait automatiquement*.

Ensuite, au niveau de la *segmentation*, notre algorithme n'est pas capable de séparer des lettres cursives et ignore totalement les espaces.

Et enfin, dans le futur, le but sera que *YOLO* reconnaisse aussi l'alphabet minuscule mais aussi l'alphabet français (avec les accents).

Bibliographie

Annexes

Figure 22: Histogramme de projection vertical

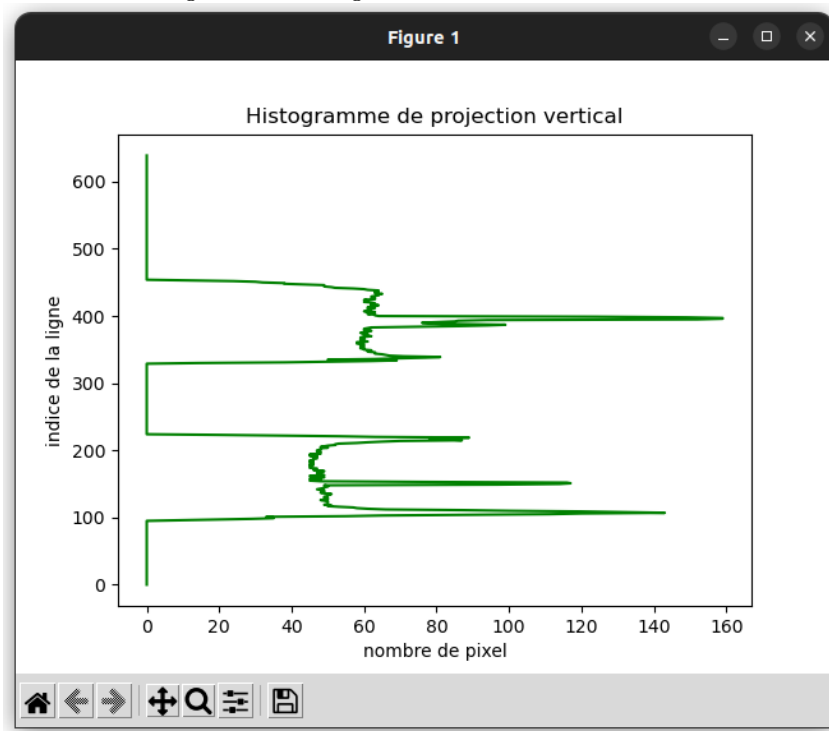


Figure 23: Histogramme de projection horizontal

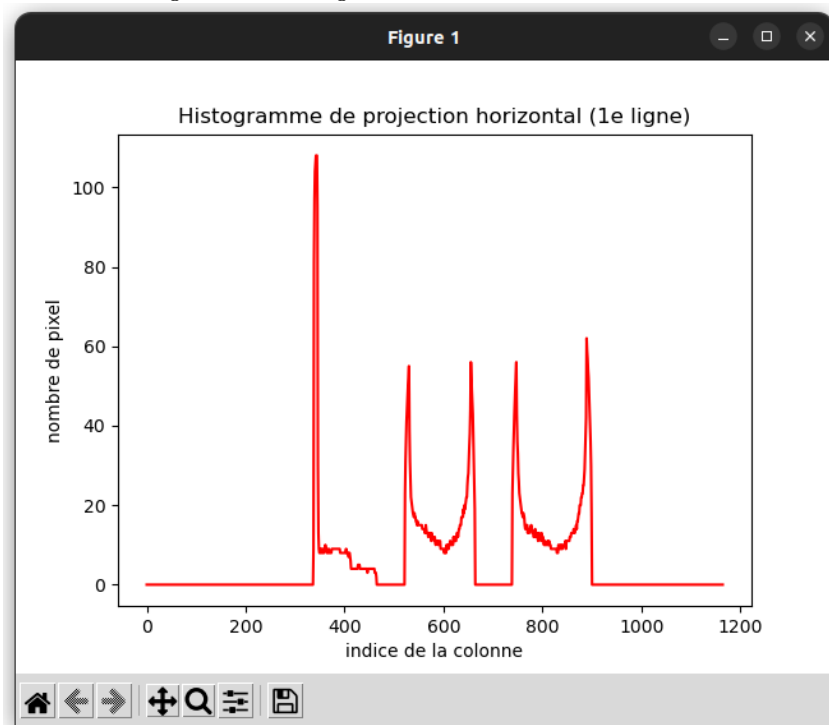


Figure 24: Histogramme de projection horizontal

