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

List of Figures

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 utilisé 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 neuronne 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.

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 classifier 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 classifier 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'entrainer un modèle au préalable. Elle est néanmoins sensible aux rotations et aux déplacements dans l'image.

1.2.3 Les réseaux de neuronnes

Les réseaux de neuronnes 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 reconnaitre 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 imagettes i.e. une image qui contient TEST devriendra 4 petites imagettes contenant respectivement T E S T.

Reconnaissance Les imagettes 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 imagettes 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 fonctinalités payantes). Nous souhaitons créer un logiciel suffisament performant pour qu'il puisse être lancé sur l'une de nos machines personelles.
De plus, une webcam est nécessaire, ou alors un appareil photo numérique.

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 accents ni caractères spécials.

2 Technologies utilisées

2.1 Langages et outils

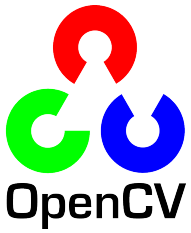
2.1.1 Python3

Python3 est un langage de programmation interprété, multiparadigme et multiplatformes. 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)`)

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 pourrons 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 archicture de Réseau Neuronal Convolutif (CNN) de type "Region based" (R-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.

Region Based CNN

YOLO

Justification

Python3

En effet, ce langage est simple à comprendre et relativement facile à lire. De plus par ça popularité, de nombreuse bibliothèques implémentes déjà ce dont nous avons besoin. Et si jamais nous rencontrons un problème, d'autres auront probablement déjà résolu le problème.

OpenCV

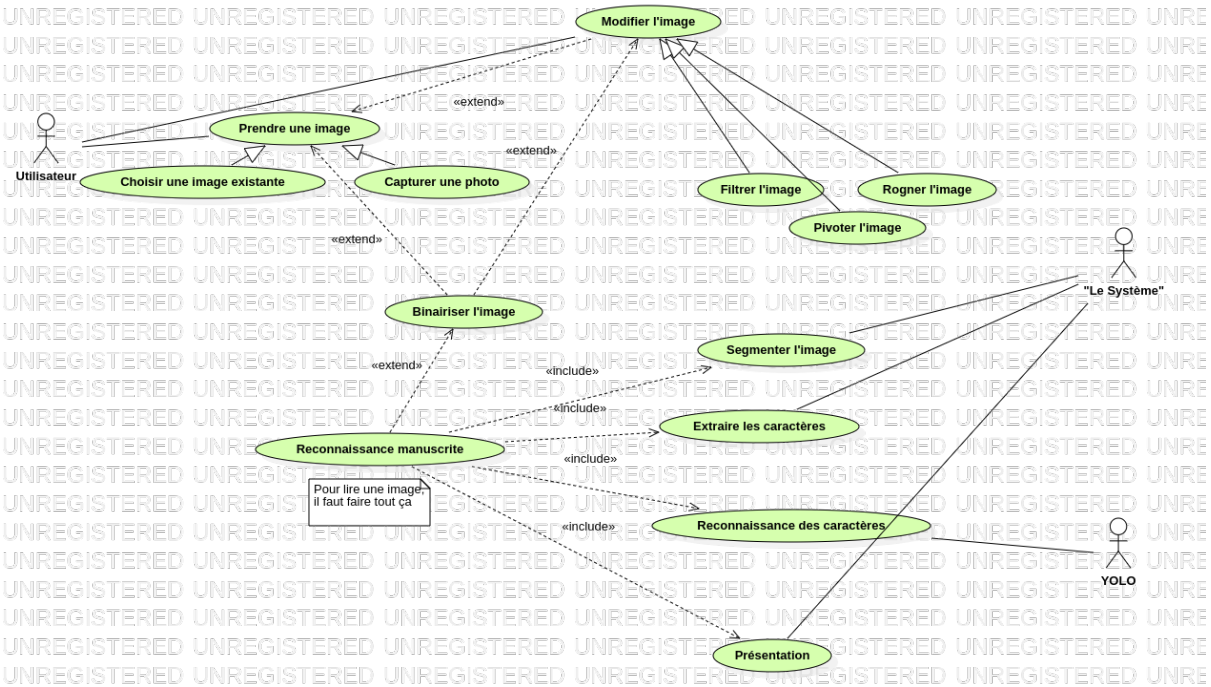
Effectivement, cette bibliothèque dispose de nombreuse fonction qui nous sont utiles. De plus, elle est populaire, sous licence libre et développé par Intel.

You Only Look Once

YOLO est un réseau de neuronne convolutif moderne est performant implément de nombreuses innovation des dernières années.

3 Développements Logiciel : Conception, Modélisation, Implémentation

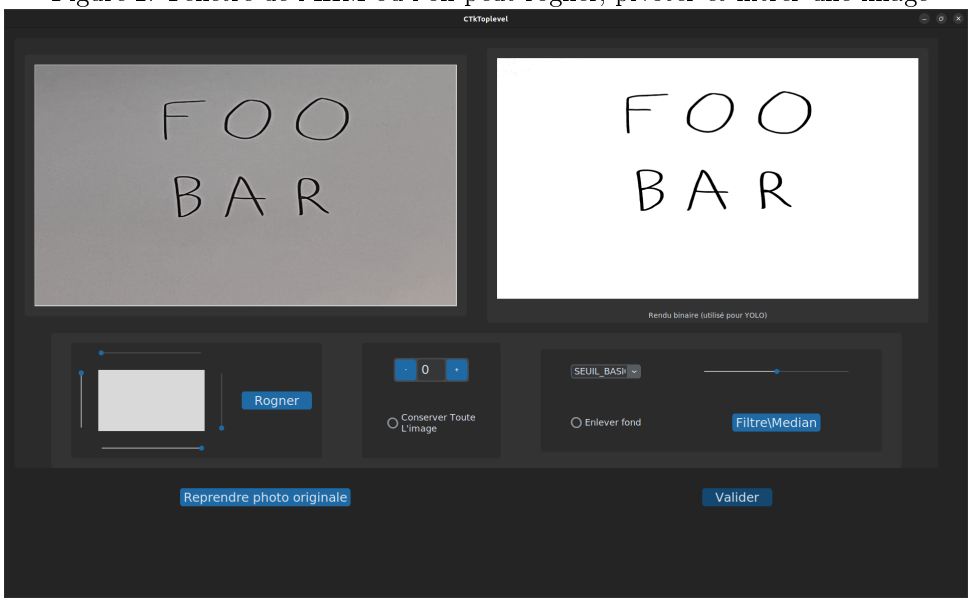
Figure 1: Diagramme de cas d'utilisation



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 entrer une image (figure 2 A), une étape de préprocessing (découpage) peut-être établi 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 facilité la segmentation des images.

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

Après avoir établi le prétraitement détaillé ci-dessus, on va procéder à la segmentation des caractères.

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 noir sur une ligne.Les images d'entrée sont ne sont pas toujours parfaite, 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 dans la partie segmentation des lignes ou des caractères. Ensuite, après segmentation des lignes et en se basant sur l'analyse de la projection verticale de l'histogramme de celles-ci, nous avions 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.

3.2 Procédures de lecture et validation des entrées

beep

3.3 Statistiques

- Nombres de lignes de code :
- Nombres de script

3.4 Entrainement de YOLO

TBD

4 Algorithmes et Analyse

4.1 Segmentation par histogramme de projection

StratStratégie 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 $projectionHorizontale = (sommeValPixel(imageBina$ 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 (x,y) dans delimitationDesLignesimagesLignes.ajouter(ndgImage[x:y, 0:longeurImage]) 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 $coordonneesCaracteres = [[]pouriallantde0longeur(im$ ()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 $imagesCaracteres[index].ajou$ $hauteurUneLigne, x : y])$ $redimensionnerImage(binarisation(imagesCaracteres[index], hauteur =$ $128, longeur = 128), seuil = 127)$ $delimitationDesLignes, delimitationDesCaracteres, imagesLignes, imagesCaracter$

Figure 4: 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) $T[i] \neq 0$!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 $tailleEspacesEntrelettres = [lettres[i+1][0]-lettres[i][1]pouride0taille(lettres)-1]$ $poidsEspace = trillageCroissant(tailleEspacesEntrelettres)$ $SommeIndice \leftarrow 0$ i de 0 à taille(poidsEspace)-1 $poidsEspace[i] \leftarrow poidsEspace[i] * (taille(poidsEspace) - i)$ $SI \leftarrow SI + i$ $moyenneEspaces \leftarrow somme(poidsEspace)/SI$ $j \leftarrow 0$ tailleEspacesEntrelettres != [] $tailleEspacesEntrelettres < moyenneEspaces*0.2$ $D = lettres[j]$; $supprimer(lettres[j])$ $F = lettres[j]$; $supprimer(lettres[j])$; inserer(tableau, indice, élément) $inserer(lettres, j, (D[0], F[1]))$ $j \leftarrow j+1$ $supprimer(tailleEspacesEntrelettres[0])$ lettres

Sur la figure 4, nous voyons une image de bonne qualité prête à être segmenter par l'algorithme n°4.1 (page 10).

Grâce à une projection vertical, nous obtenons ce tableau (voir figure 6). 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 5.

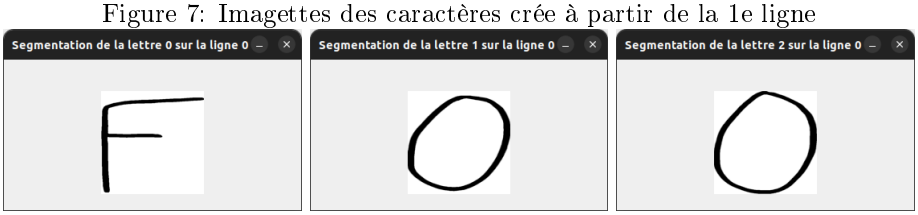
Figure 6: Structure de données dans l'algorithme SEGMENTATION

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

Maintenant que nous avons isoler les lignes, nous allons maintenant découper les lettres de chaque lignes en suivant l'algorithme n°4.1 page 11

Figure 5: Images de nos 3 lignes crée à partir de l'image de départ

Figure 7: Imagettes des caractères crée à partir de la 1e ligne



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 dectecter 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 6 et des histogrammes de projection horizontal dans l'annexe (page 16).

Figure 8: Imagettes des caractères crée à partir de la 2e ligne

5 Analyse des Résulats

5.1 Mesure expérimental de YOLO

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

Figure 9:

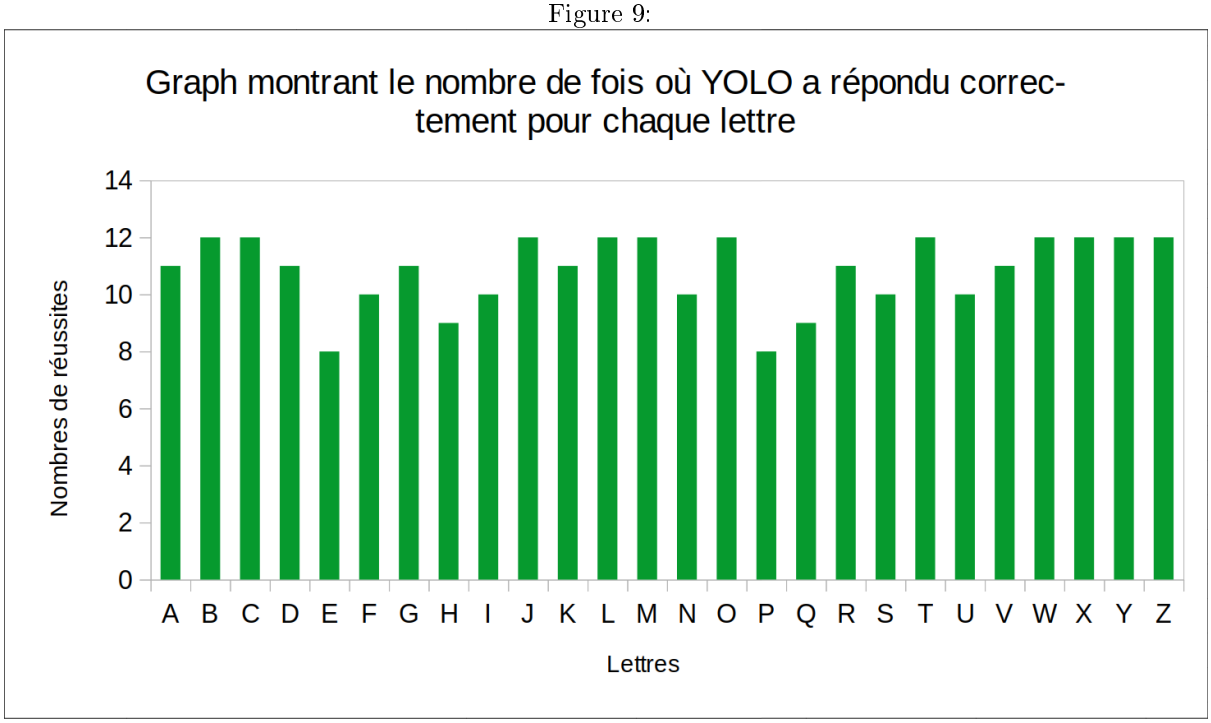
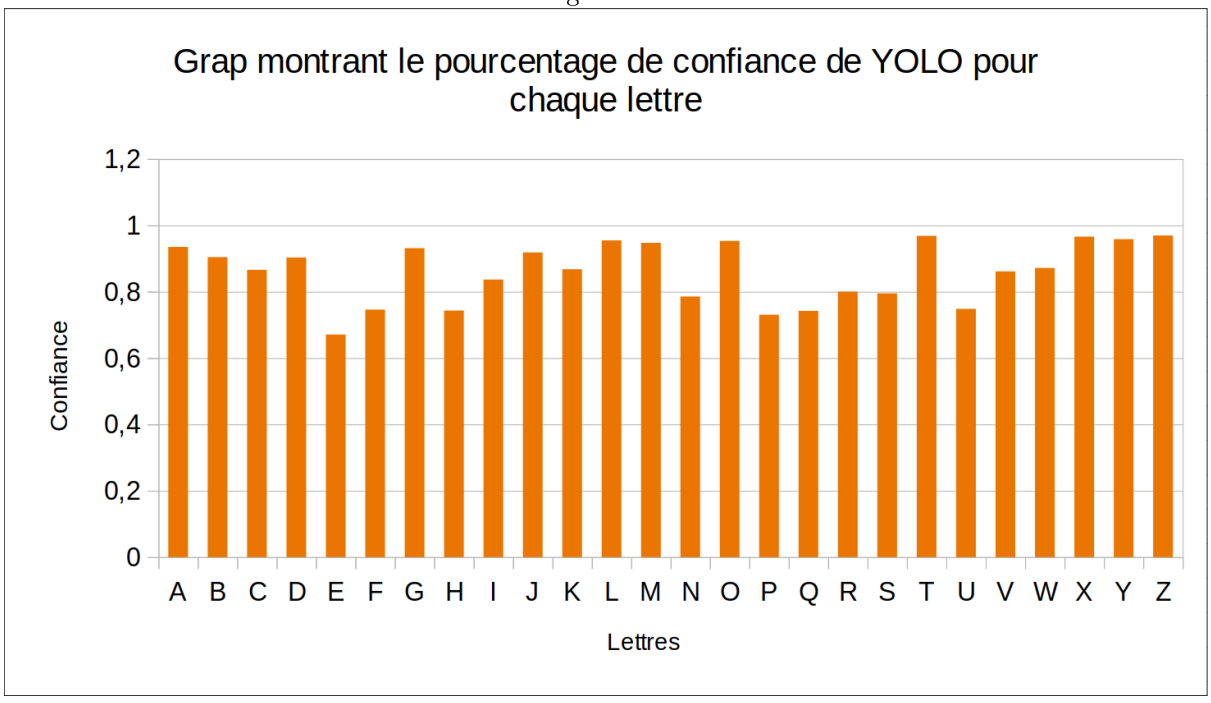


Figure 10:



6 Gestion du Projet

Pendant ce projet, nous avons due communiquer et collaborer ensembles afin d'atteindre not but commun. Voici les outils que nous avons utilisé.

Premierement, pour communiquer nous avons utiliser 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 *rognagne soit fait automatiquement*.

Ensuite, au niveau de la *segmentation*, notre algorithme n'est pas capable de séparer des lettre 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 11: Histogramme de projection vertical

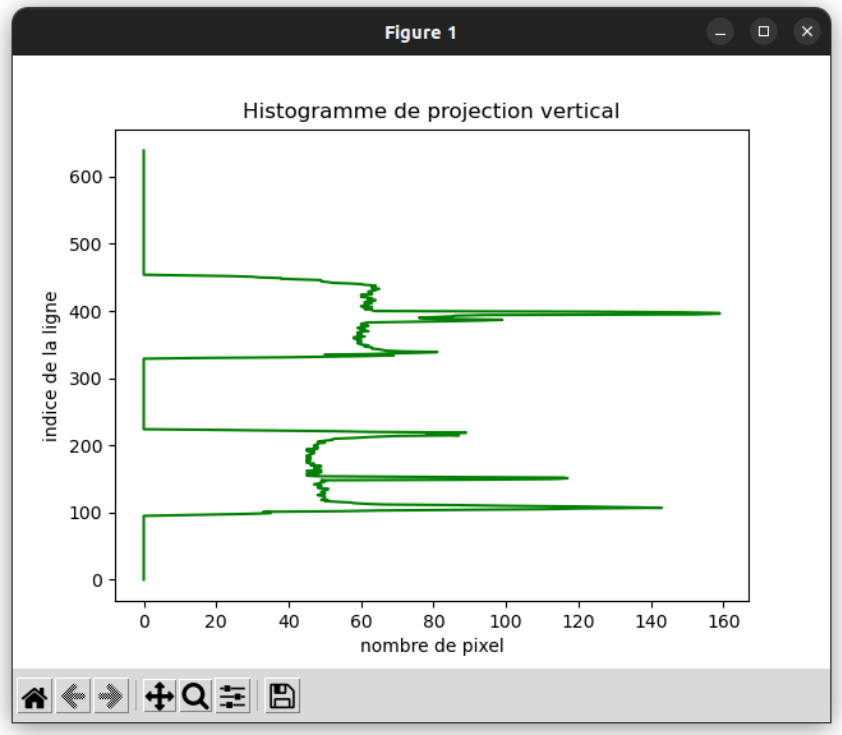


Figure 12: Histogramme de projection horizontal

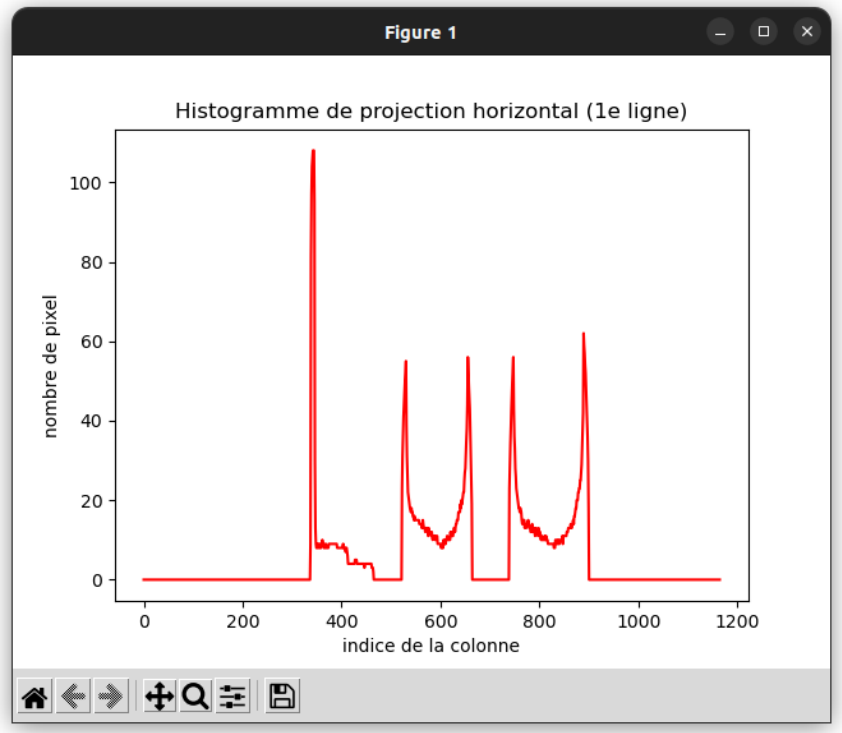


Figure 13: Histogramme de projection horizontal