

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

1	Présentation du Sujet	4
1.1	La problématique	4
1.2	Quelques approches de reconnaissance de formes	4
1.2.1	k-Nearest Neighbor	4
1.2.2	Template matching	4
1.2.3	Les réseaux de neurones	4
1.3	Cahier des charges	4
1.3.1	Objectifs	4
1.3.2	Besoins et contraintes	5
1.3.3	Résultats attendus	5
2	Technologies utilisées	5
2.1	Langages et outils	5
2.1.1	Python3	5
2.1.2	OpenCV	5
2.1.3	You Only Look Once	6
3	Développements Logiciel : Conception, Modélisation, Implémentation	7
3.1	Interface Homme-Machine	7
3.2	Procédures de lecture et validation des entrées	8
3.3	Statistiques	8
3.4	Entraînement de YOLO	9
4	Algorithmes et Analyse	9
4.1	Segmentation par histogramme de projection	9
5	Analyse des Résultats	14
5.1	Mesure expérimental de YOLO	14
6	Gestion du Projet	15
	Bilan et Conclusions	15

Table des figures

1	Diagramme de cas d'utilisation	7
2	Fenêtre de l'IHM où l'on peut rogner, pivoter et filtrer une image	7
3	Fenêtre de l'IHM après présentation des caractères reconnus	8
4	Exemple d'une image après filtrage	11
6	Structure de données dans l'algorithme SEGMENTATION	11
5	Images de nos 3 lignes crée à partir de l'image de départ	11
7	Imagettes des caractères crée à partir de la 1e ligne	13
8	Imagettes des caractères crée à partir de la 2e ligne	13
9	14
10	14
11	Histogramme de projection vertical	17
12	Histogramme de projection horizontal	17
13	Histogramme de projection horizontal	18

List of Algorithms

1	SEGMENTATION(image)	10
2	COORDONNEECARACTERE(T)	12

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 compte tenu des enjeux actuels. Notamment, vis à vis du développement des IAs étant très utilisées par les entreprises sachant le gain de temps et de productivité considérable que ces dernières peuvent leur apporter.

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. Puis lorsqu'une nouvelle image devra être classifiée, une simple comparaison entre cette image et les images de référence sera faite à l'aide d'une 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 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. Mais il pourra également importer des images depuis son système de fichier. Tout cela à travers une interface Homme-Machine.

Pré-traitement Le logiciel devra améliorer la qualité de l'image capturée en appliquant une réduction bruit et l'utilisation de 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 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 multiplateforme. 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)))

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 Neuronale Convolutif (CNN) de type "Region based" (R-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.

Region Based CNN

YOLO

Justification

Python3

Nous avons choisi ce langage car il est simple à comprendre et relativement facile à lire. De plus, par sa popularité, il existe de nombreuses bibliothèques dont les fonctions répondent à nos besoins. Python étant conçu pour faciliter la programmation, nous pourrions nous concentrer davantage sur le développement de nos algorithmes et être efficace quant à leur implémentation.

OpenCV

Effectivement, cette bibliothèque dispose de nombreuses fonctions qui nous sont utiles. De plus, elle est populaire, sous licence libre et développée par Intel.

You Only Look Once

YOLO est un réseau de neurone convolutif moderne et performant implémenté de nombreuses innovations des dernières années.

3 Développements Logiciel : Conception, Modélisation, Implémentation

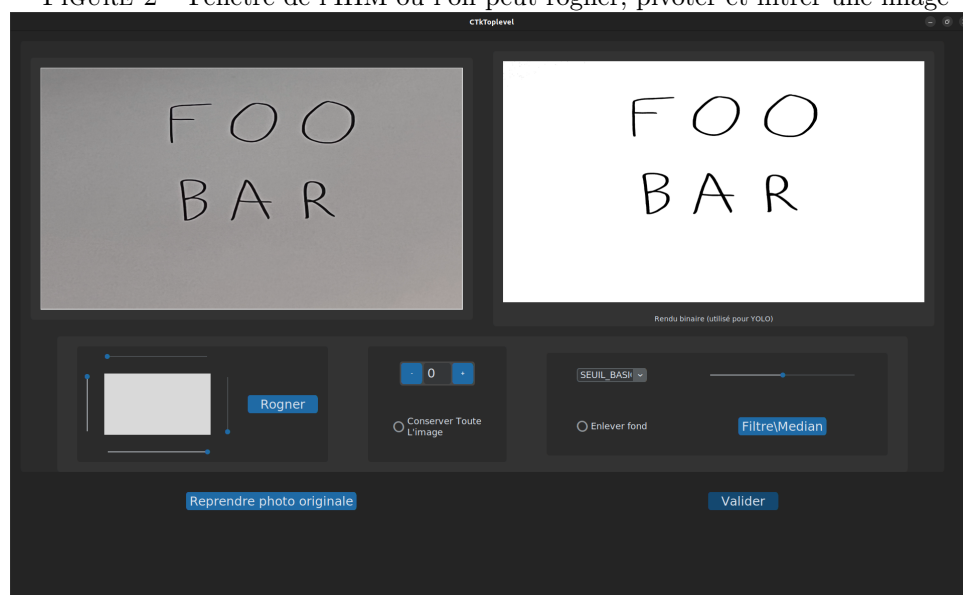
FIGURE 1 – Diagramme de cas d'utilisation



3.1 Interface Homme-Machine

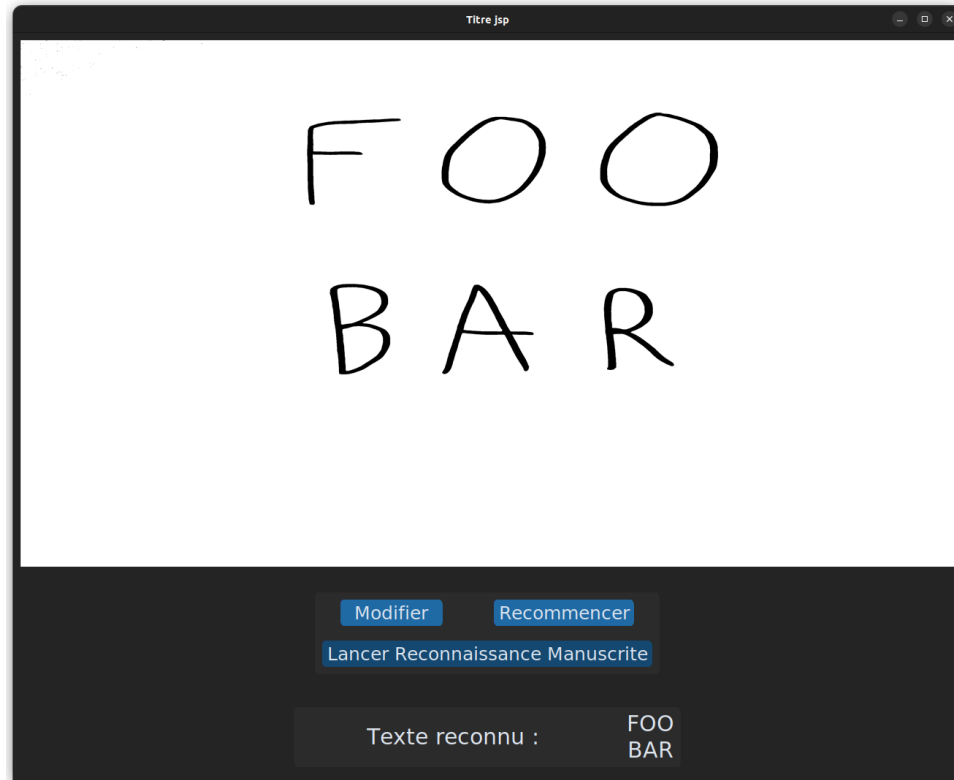
L'IHM nous permet de prendre des photos avec une webcam connectée à l'ordinateur ou choisir à partir d'une fenêtre une image déjà existante.

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



Après avoir 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 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 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 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 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 obtenons 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 Entraînement de YOLO

TBD

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'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

```
1 ndgImage ← convertirImageEnNDG(image);
   /* on met l'image en noir et blanc sauf que le seuillage est inversé */
2 imageBinaire ← binarisationInvers(ndgImage);
   /* on fait la somme des pixels sur les colonnes et on les divise par 255 */
3 projectionHorizontale = (sommeValPixel(imageBinaire, axe = 1))/255;
4 delimitationDesLignes = coordonneeLigne(projectionHorizontale);
5 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 */
6 for (x,y) dans delimitationDesLignes do
7   | imagesLignes.ajouter(ndgImage[x :y, 0 :longeurImage])
8 end
   /* on initialise le tableau de tableaux des images de Caractère on fonction du
   nombre de ligne détecté */
9 imagesCaracteres = [[]pouriallantde0longeur(imagesLignes)];
   /* on initialise le tableau de tableaux de couple de coordonnées on fonction du
   nombre de ligne */
10 coordonneesCaracteres = [[]pouriallantde0longeur(imagesLignes)];
11 for index, uneLigne dans imagesLignes do
12   | ligneBinaire = binarisationInvers(uneLigne);
   /* on fait la somme des pixels sur les lignes de l'image et on les divise par
   255 */
13   | projectionVerticale = (sommeValPixel(imageBinaire, axe = 0))/255;
14   | delimitationDesCaracteres = coordonneeCaractere(projectionVerticale);
   /* on ajoute les coordonnées au tableau de coordonnées */
15   | 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 */
16   | for (x,y) dans delimitationDesCaracteres do
17     | imagesCaracteres[index].ajouter(ndgImage[0 : hauteurUneLigne, x : y]);
18   | end
19   | redimensionnerImage(binariation(imagesCaracteres[index], hauteur = 128, longueur =
   128), seuil = 127);
20 end
21 return delimitationDesLignes, delimitationDesCaracteres, imagesLignes, imagesCaracteres
```

FIGURE 4 – Exemple d’une image après filtrage



Sur la figure 4, nous voyons une image de bonne qualité prête à être segmenter par l’algorithme n°1 (page10).

Grâce à une projection vertical, nous obtenons ce tableau (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. Ensuite, on extrait nos 2 lignes comme sur la figure 5 depuis l’image d’entrée.

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)]
```

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

FIGURE 5 – Images des deux lignes extraites à partir de l’image de départ



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

```
1 coordCaractere = [];  
2 dansLeCaractere = False;  
   /* Début = pair Fin = impair */  
3 DF = [];  
4 for i de 0 à taille(T) do  
5   if T[i] ≠ 0 then  
6     if !dansLeCaractere or (dansLeCaractere and i == taille(T)-1) then  
7       DF.ajouter(i);  
8       if i == taille(myprojection)-1 et taille(DF) mod 2 ≠ 0 then  
9         | supprimer(DF[taille(DF) - 1]);  
10      end  
11    end  
12    if i == len(myprojection)-1 and len(coordDF)coordDF.pop(-1)  
13      dansLeCaractere = True;  
14  end  
15  else  
16    if dansLeCaractere then  
17      | DF.ajouter(i)  
18    end  
19    dansLeCaractere = False;  
20  end  
21 lettres ← [ (DF[i],DF[i+1]) pour i de 0 à taille(DF) avec un pas de 2 ];  
22 if taille(lettres)>1 then  
23   tailleEspacesEntrelettres = [lettres[i + 1][0] - lettres[i][1]pouride0taille(lettres) - 1];  
24   poidsEspace = trillageCroissant(tailleEspacesEntrelettres);  
25   SommeIndice ← 0;  
26   for i de 0 à taille(poidsEspace)-1 do  
27     | poidsEspace[i] ← poidsEspace[i] * (taille(poidsEspace) - i);  
28     | SI ← SI + i;  
29   end  
30   moyenneEspaces ← somme(poidsEspace)/SI;  
31   j ← 0;  
32   while tailleEspacesEntrelettres != [] do  
33     if tailleEspacesEntrelettres < moyenneEspaces * 0.2 then  
34       D = lettres[j]; supprimer(lettres[j]);  
35       F = lettres[j]; supprimer(lettres[j]);  
36       /* inserer(tableau, indice, élément) */  
37       inserer(lettres, j, (D[0], F[1]));  
38     end  
39     else j ← j + 1;  
40     supprimer(tailleEspacesEntrelettres[0]);  
41   end  
42 return lettres
```

FIGURE 7 – Imagettes des caractères créées à partir de la 1e ligne

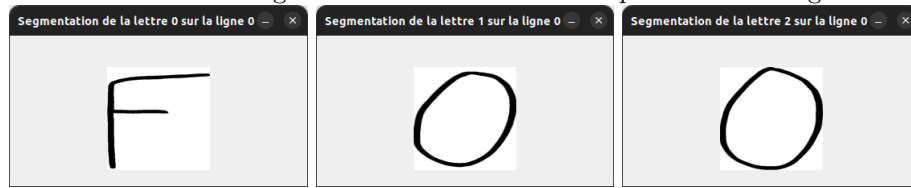
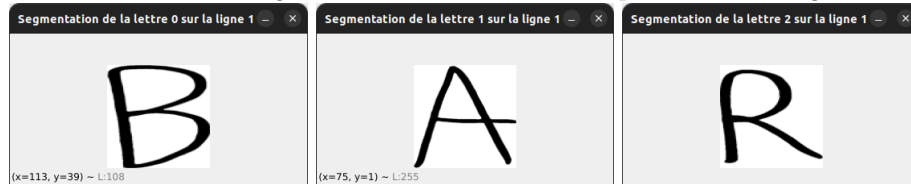


FIGURE 8 – Imagettes des caractères créées à partir de la 2e ligne



page 12

Étant donné que l'image est binarisée, lorsqu'un caractère est présent à un endroit donné, cela signifie que sur la projection verticale de l'histogramme est supérieur à 0 à cette position. L'algorithme n°2 va parcourir cette projection de l'histogramme, détecter 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 résultats de cette fonction sur la figure 6 et des histogrammes de projection horizontal dans l'annexe (page 17).

5 Analyse des Résultats

5.1 Mesure expérimental de YOLO

Notre procédure de test est la suivante : Écrire une suite de caractère de moins en moins bien écrit pour chaque lettre de l'alphabet, ???, YOLO, graph

FIGURE 9 –

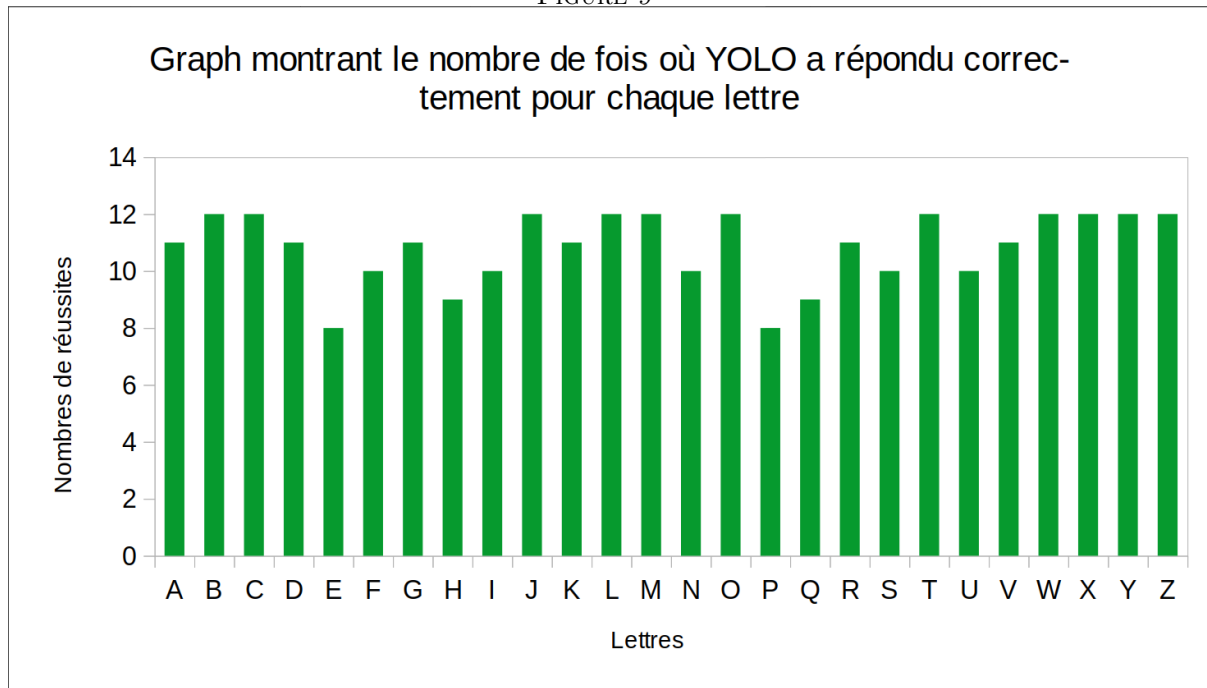
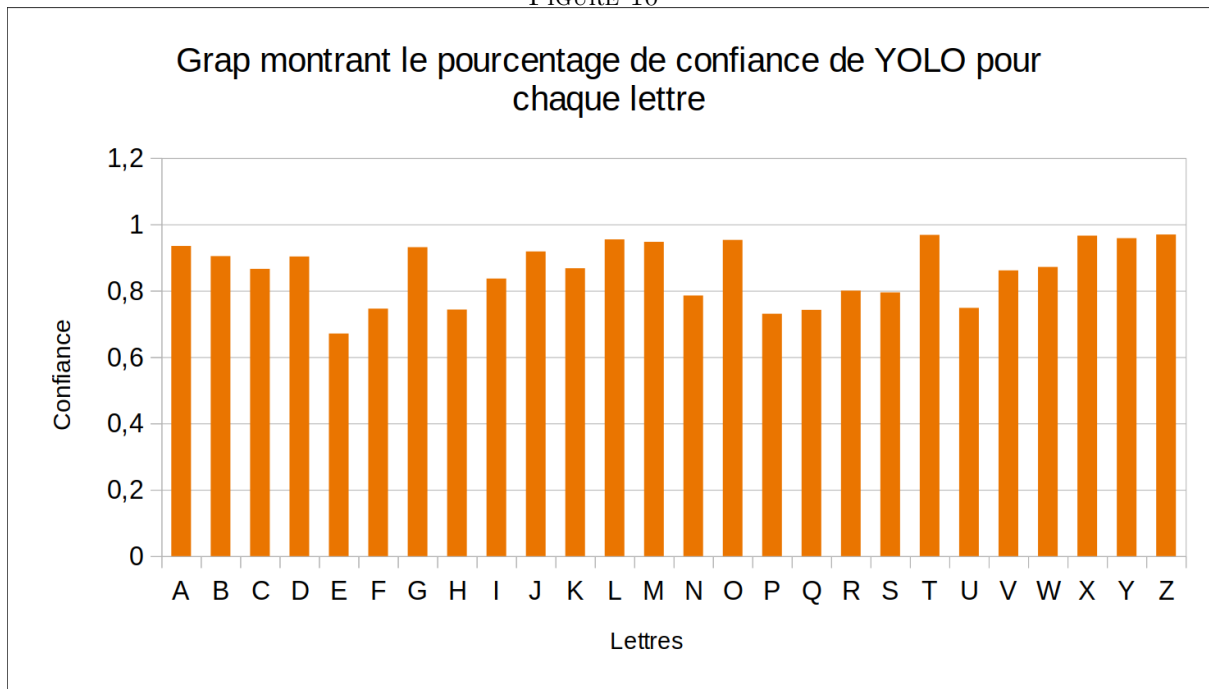


FIGURE 10 –



6 Gestion du Projet

Pendant ce projet, nous avons dû communiquer régulièrement afin d'être au point sur les éléments à développer ou corriger, nécessitant ainsi l'utilisation de Discord.

De plus, pour le partage de code et le versioning, nous avons utilisés git et Github
 Laurent : filtrage, segmentation
 Romain : IHM, YOLO
 Tony : rapport, segmentation

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 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

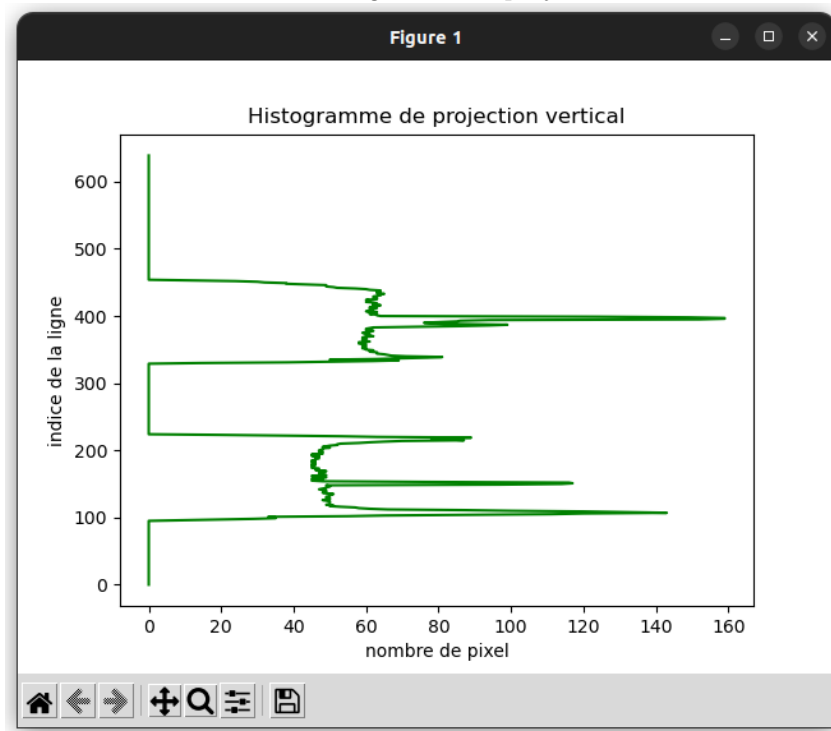


FIGURE 12 – Histogramme de projection horizontal

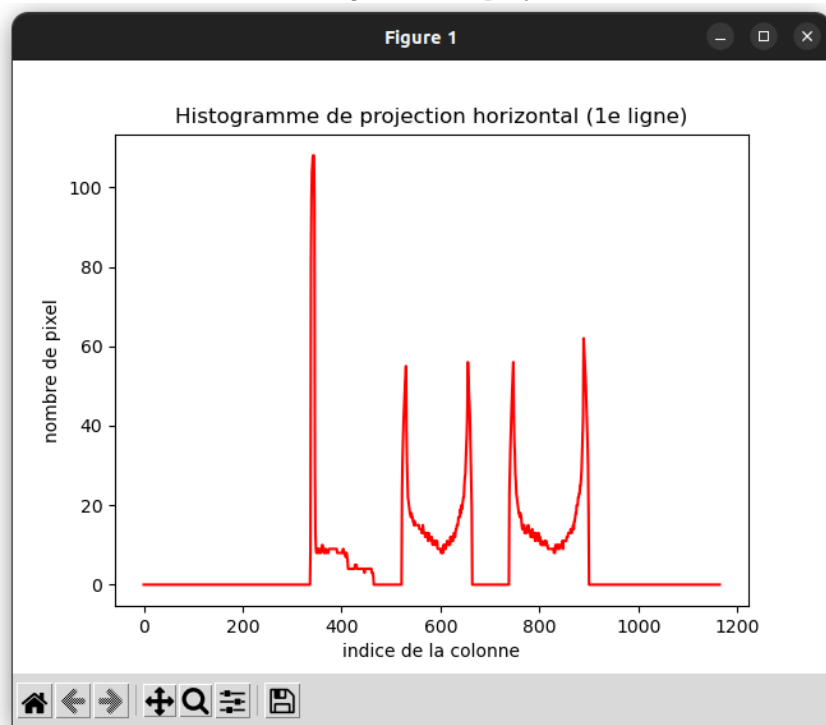


FIGURE 13 – Histogramme de projection horizontal

