

Reconnaissance de l'écriture manuscrite

Rapport de projet

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

11 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	Objectifs	4
1.4	Cahier des charges	5
1.4.1	Besoins et contraintes	5
1.4.2	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 basé sur les méthodes de réseaux de neurones qui ont prouvé leur efficacité à reconnaître et classer les formes les plus complexes.

Cette recherche va nous permettre de mettre en œuvre nos compétences en traitement d'image et d'explorer de nouvelles méthodes plus avancées.

Ce problème est particulièrement intéressant. En effet, un projet sur le traitement automatique de l'information numérique est exactement ce que des étudiants en informatique devraient faire. Nous avons pu constater une explosion de l'intelligence artificielle ces dernières années. Notamment les réseaux neuronaux. À l'heure actuelle, ils sont capables de détecter des objets et de les classer. Ce projet est tout à fait intéressant pour de futurs étudiants en master IASD.

1.2 Quelques approches de reconnaissance de formes

1.2.1 k-Nearest Neighbor

Méthode statistique, il y a un plan avec plein d'objets qui forment des troupes, notre lettre est quelque part dans le plan, on trace un cercle autour, notre lettre est identifiée comme l'objet présent au plus grand nombre à l'intérieur du cercle.

1.2.2 Template matching

Cette approche est l'une des plus simples. Voici en quoi elle consiste :
Nous allons partir d'une banque d'images de référence. À chaque fois que nous souhaitons classer une nouvelle image, nous allons comparer cette nouvelle image avec toutes nos images de référence à l'aide d'une simple fonction de distance euclidienne.

L'avantage de cette solution est que nous n'avons pas besoin d'entraîner un modèle au préalable.

Le désavantage est que cette technique est sensible aux rotations et aux déplacements dans l'image.

1.2.3 Les réseaux de neurones

Les réseaux de neurones

1.3 Objectifs

Notre objectif est de proposer une méthode pour convertir une image de texte manuscrit en un texte numérique.

Dans notre étude nous avons considéré 26 lettres de l'alphabet latin en majuscules et présenter des résultats préliminaires

1.4 Cahier des charges

1.4.1 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 imageries .i.e une image qui contient TEST deviendra 4 petites imageries contenant respectivement T E S T.

Reconnaissance Les imageries 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 imageries 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.

1.4.2 Résultats attendus

Notre programme doit reconnaître des lettres manuscrites sur un fond blanc. Les lettres seront des caractères majuscule non-liées et sans accents ni caractère spécial.

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)))

2.1.2 OpenCV

Dans ce projet, nous choisissons d'utiliser la librairie open source de vision par ordinateur (Open Computer Vision). Elle contient les fonctionnalités de pré-traitement d'image nécessaire dans ce projet. Nous utiliserons surtout les fonctions de filtrage d'image.



2.1.3 You Only Look Once

Expliquons (un peu tout petit peu) plus en détail les réseaux de neurones convolutifs Les réseaux de neurone simulent plus ou moins le fonctionnement de notre cerveau. Parlons d'abord du neurone formel et on va progresser jusqu'aux CNN.

Le neurone va prendre plusieurs entrées, des nombres, faire une opération avec une fonction d'activation et en sortie retourner le résultat.

Réseau de neurone L'architecture des réseaux de neurone simple est organisée en couche. La 1er couche qui est l'entrée (censée représenter les yeux). La dernière couche représente la sortie, par exemple si le réseau est entraîné pour reconnaître des chiens, la sortie doit retourner une valeur proche de 1 si c'est un chien et 0 sinon. 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 suivante mais pas à ceux de la même couche.

Le désavantages de cette approche est qu'il est nécessaire de lui donner des "caractéristiques". Pour reconnaître si cet animal est un chien on ne va pas lui donner l'image directement, on va lui dire en entrée si ça a des poils, la couleur de son pelage, le nombre de pattes, la forme des oreilles.

Les CNN blablabla

R-CNN

You Only Look Once (YOLO) est une architecture 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, les classifier.

Justification

Python3

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

OpenCV

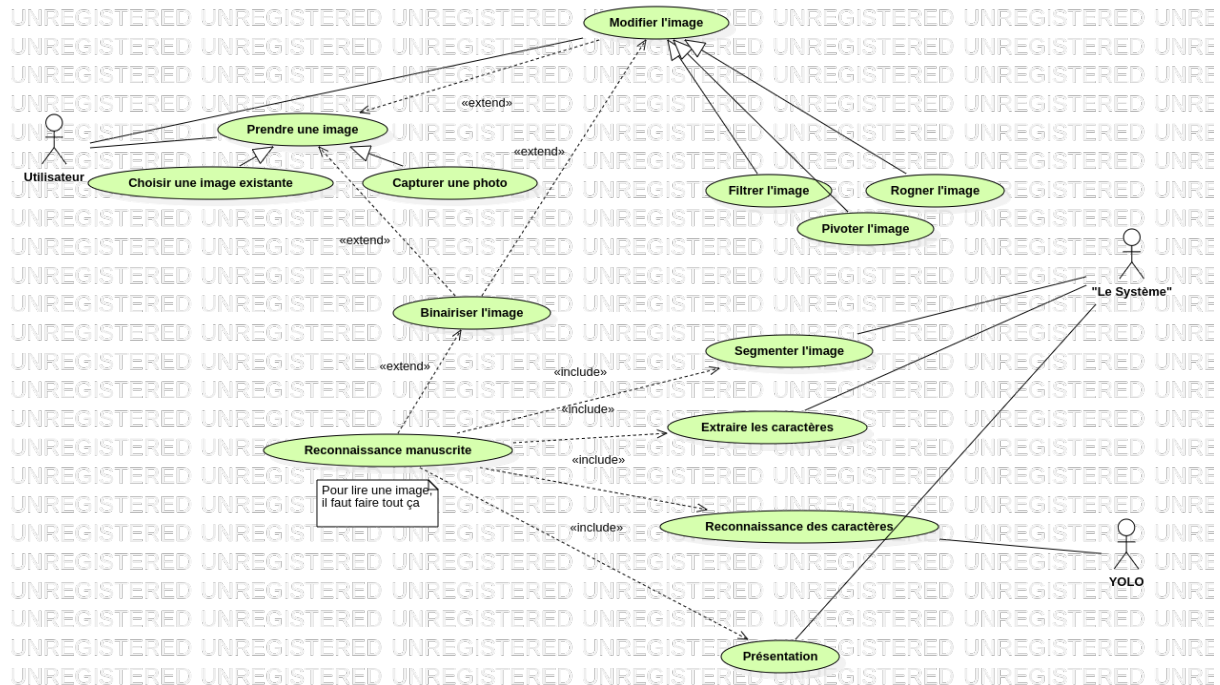
Effectivement, cette librairie dispose de nombreuses fonction qui nous sont utiles. De plus, elle est populaire, sous licence libre et développé par Intel.

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

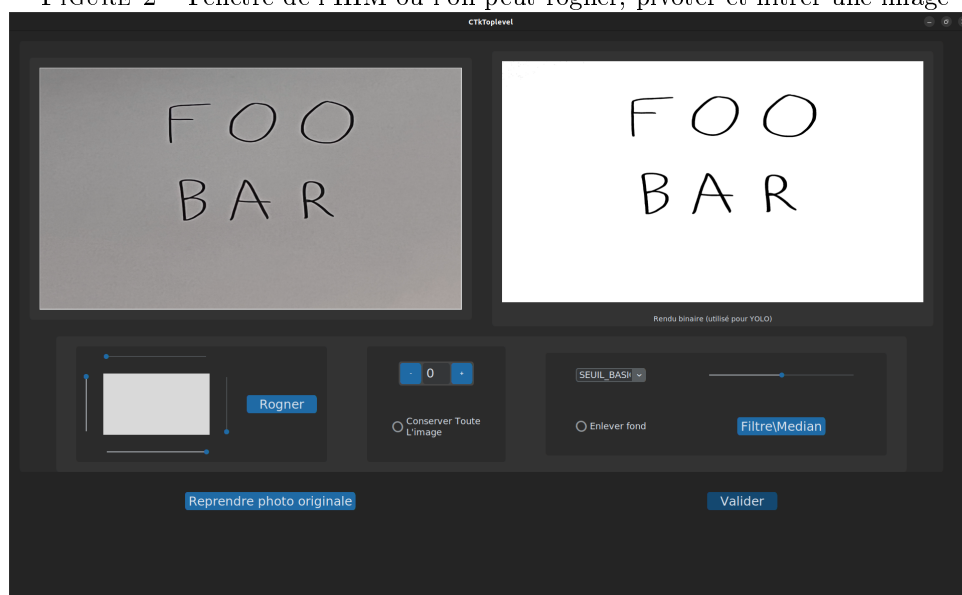
FIGURE 1 – Diagramme de cas d'utilisation



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

Dans un premier temps, une projection d'histogramme vertical de l'image binaire est calculée ; cette projection d'histogramme vertical correspond à un vecteur de taille égale à la largeur de l'image et sommant les pixels blancs et noirs par colonne. Les images d'entrée sont généralement bruitées, ce qui provoque l'apparition de quelques petits pics indésirables dans l'image de l'histogramme de projection. Afin de supprimer cet effet de bruit et de supprimer ces pics, nous avons créé une fonction de seuillage. Ensuite, et en se basant sur l'analyse de l'image de projection de l'histogramme vertical, nous avons fait écrire une fonction pour compter le nombre de caractères et les localiser. Le critère de détection est basé sur la succession de projections nulles et non nulles. En sortie, nous avons le nombre de caractères et les coordonnées (début, fin) de chacun. Par conséquent, nous implémentons la fonction qui réalise la segmentation du mot entier.

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

Algorithm 1: SEGMENTATION(image)

Data: image en RGB

Stratégie: D'abord, une segmentation des lignes par projection vertical. Puis, dans un 2e temps, segmentation des caractères grâce à une projection horizontal de chaque ligne individuellement

Result:

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

Tableau de tableau de couple d'entier représentant les coordonnée x de début et de fin des lettres par lignes,

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 un caractère grâce aux
      coordonnées calculées */
   /* et on l'ajoute à notre tableau d'image des 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 detecté */
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 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(imagesCaracteres[index], hauteur = 128, longueur = 128);
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 (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)]
[]
```

FIGURE 5 – Images de nos 3 lignes crée 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     end  
9     dansLeCaractere = True;  
10  end  
11  else  
12    if dansLeCaractere then  
13      DF.ajouter(i)  
14    end  
15    dansLeCaractere = False;  
16  end  
17 end  
18 lettres ← [ (DF[i],DF[i+1]) pour i de 0 à taille(DF) avec un pas de 2 ];  
19 tailleEspacesEntrelettres = [lettres[i+1][0] - lettres[i][1] pour i de 0 à taille(lettres) - 1];  
20 poidsEspace = trillageCroissant(tailleEspacesEntrelettres);  
21 SommeIndice ← 0;  
22 for i de 0 à taille(poidsEspace)-1 do  
23   poidsEspace[i] ← poidsEspace[i] * (taille(poidsEspace) - i);  
24   SI ← SI + i;  
25 end  
26 moyenneEspaces ← somme(poidsEspace)/SI;  
27 j ← 0;  
28 while tailleEspacesEntrelettres != [] do  
29   if tailleEspacesEntrelettres < moyenneEspaces * 0.2 then  
30     D = lettres[j]; supprimer(lettres[j]);  
31     F = lettres[j]; supprimer(lettres[j]);  
    /* inserer(tableau, indice, élément) */  
32     inserer(lettres, j, (D[0], F[1]));  
33   end  
34   else j ← j + 1;  
35   supprimer(tailleEspacesEntrelettres[0]);  
36 end  
37 return lettres
```

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

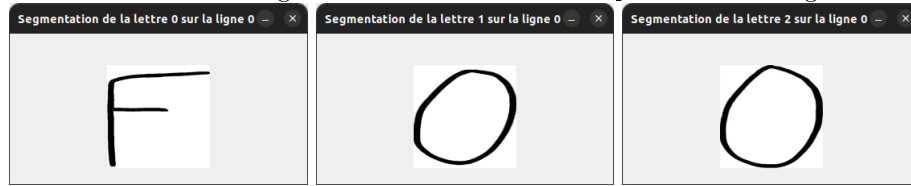
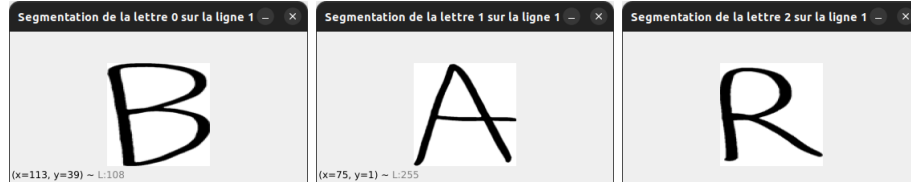


FIGURE 8 – Imagettes des caractères crée à partir de la 2e ligne



Maintenant que nous avons isoler les lignes, nous allons maintenant découper les lettres de chaque lignes en suivant l'algorithme n°2 page 12

Étant donnée que l'image est filtré, en noir et blanc et binarisé, 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°2 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 6 et des histogrammes de projection horizontal dans l'annexe (page 17).

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 9 –

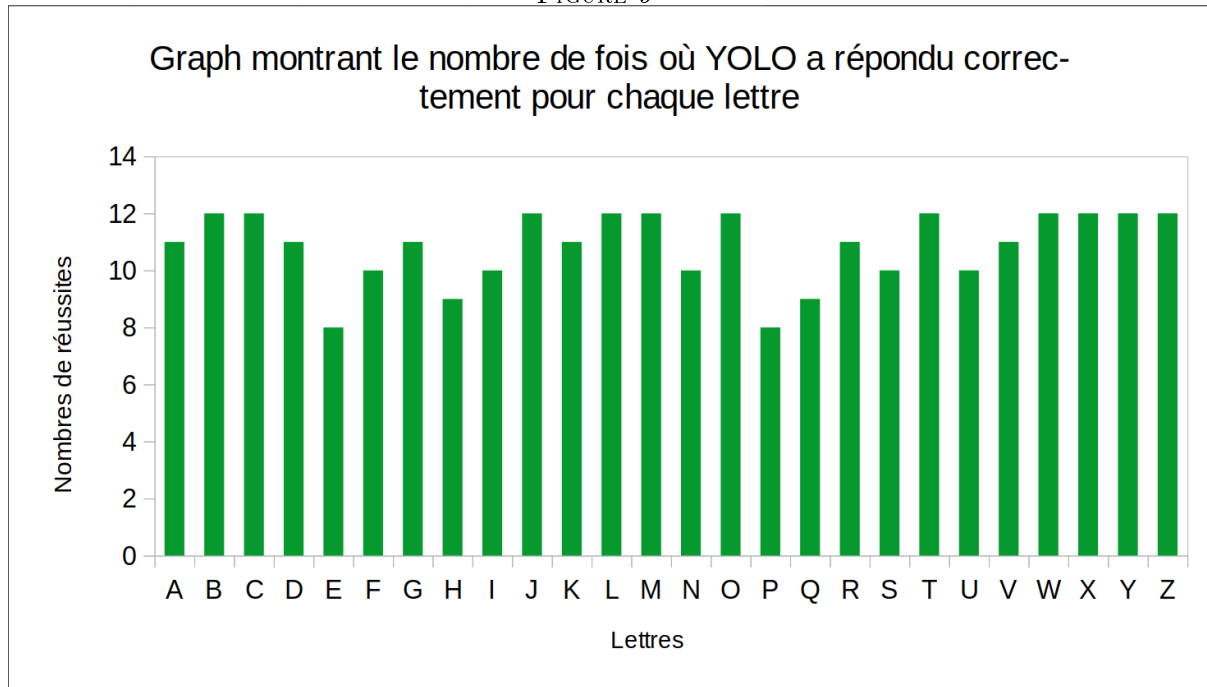
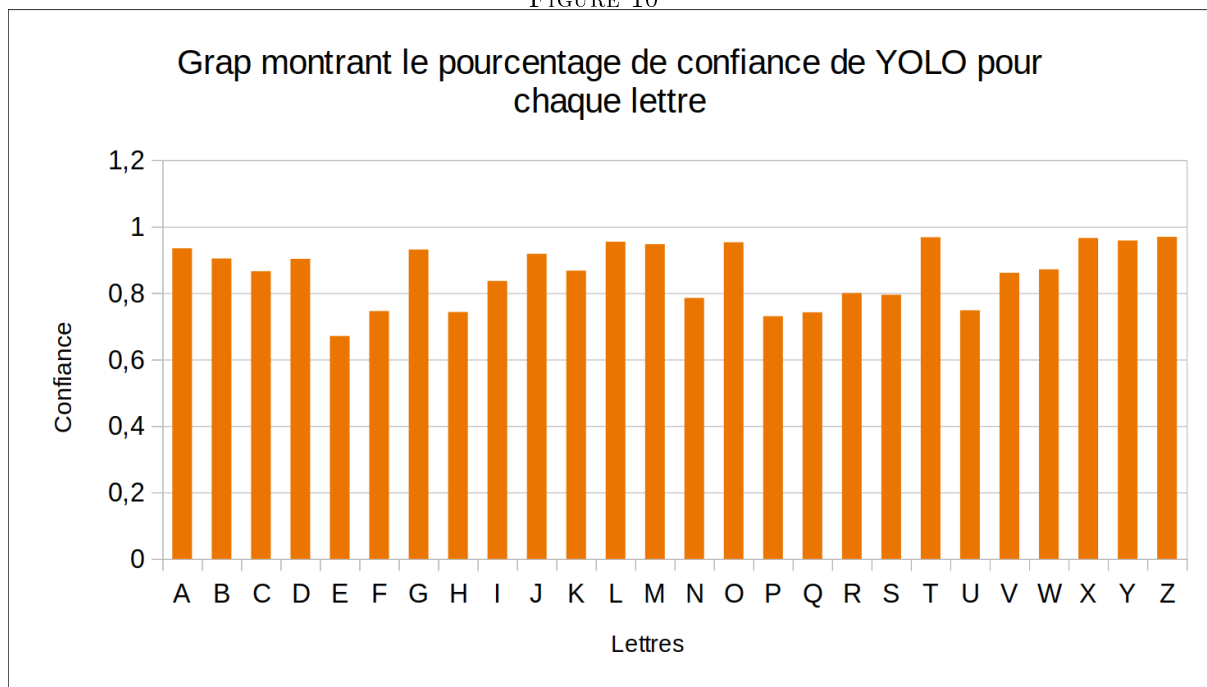


FIGURE 10 –



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 11 – Histogramme de projection vertical

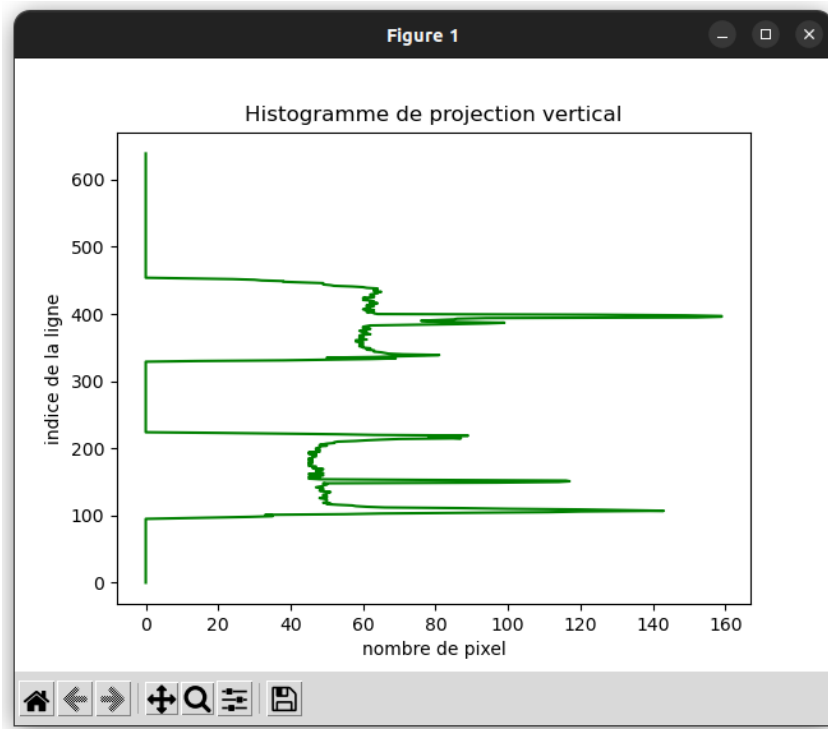


FIGURE 12 – Histogramme de projection horizontal

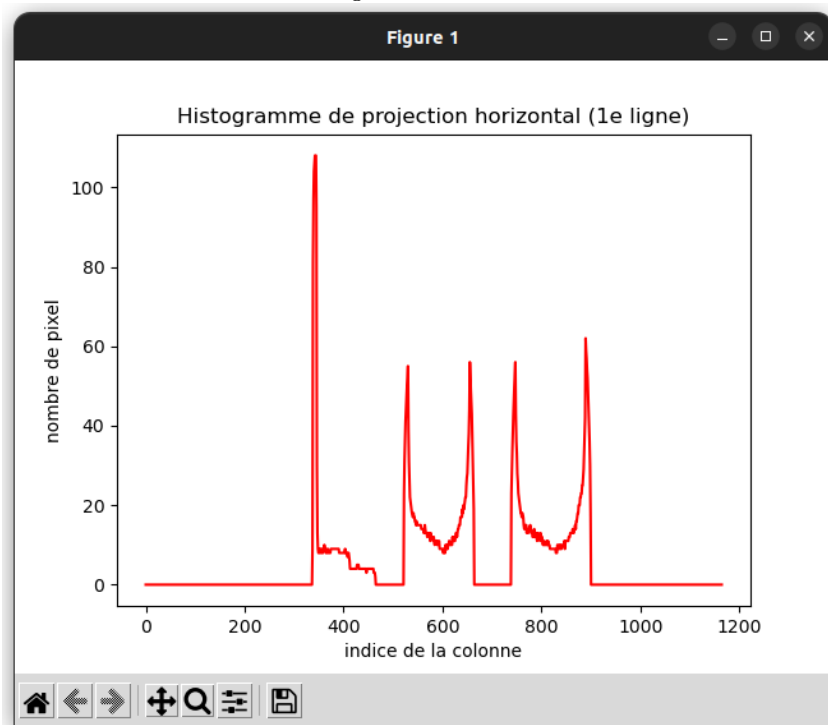


FIGURE 13 – Histogramme de projection horizontal

