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

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. Bien que cette application est utilisé dans divers domaine, la reconnaissance de l'écriture manuscrite est toujours considéré comme un problème? ? ?. La variation des styles d'écritures manuscrites d'une personne à l'autre et la mauvaise qualité du texte pose des obstacles importants à sa convertion en texte numérique.

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 actuel, ils sont capables de detecter des objets et de les classifier. Ce projet est tout à fait intéressant pour de futurs étudiants en master IASD.

1.2 Les différentes approches

1.2.1 k-Nearest Neighbor

Méthode statistique, il y a un plan avec plein d'objets qui forment des troupeaux, 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 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 comparé 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 que nous n'avons pas besoin d'entrainer un modèle au préalable.

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

1.3 Objectifs

Notre but est de créer un logiciel pour tranformer une image qui contient des symboles en un texte numérique.
Initialement, nous nous concentrerons sur des images simples et de bonne qualité avec des symboles de l'alphabet latin (26 lettres) seulement en majuscules sur 1 ligne. Puis, si le temps nous le permet, on aggrandira la portée du problème.

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 charactères présents sur l'image seront localisés à l'aide d'histogramme de projection.

Extraction des charactères Les charactères 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 charactère. 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.

1.4.2 Résulats attendus

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

2 Technologies utilisées

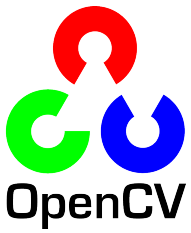
2.1 Langages et outils

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)`)

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 reprensenter les yeux). La dernière couche représente la sortie, par exemple si le réseau est entrainé pour reconnaitre 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 "charactéristiques". Pour reconnaitre 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 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, les classifier.

Justification

Python3

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

OpenCV

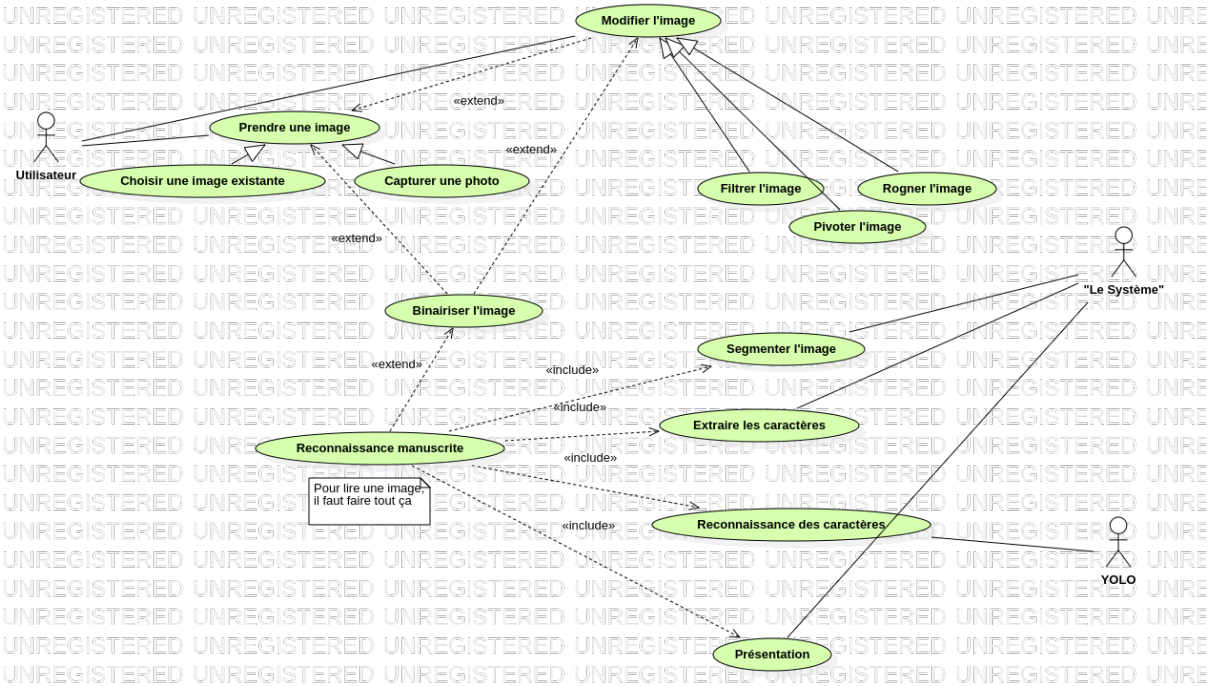
Effectivement, cette librairie dispose de nombreuse fonction qui nous sont utiles. De plus, elle est populaire, sous licence libre et développé par Intel.

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

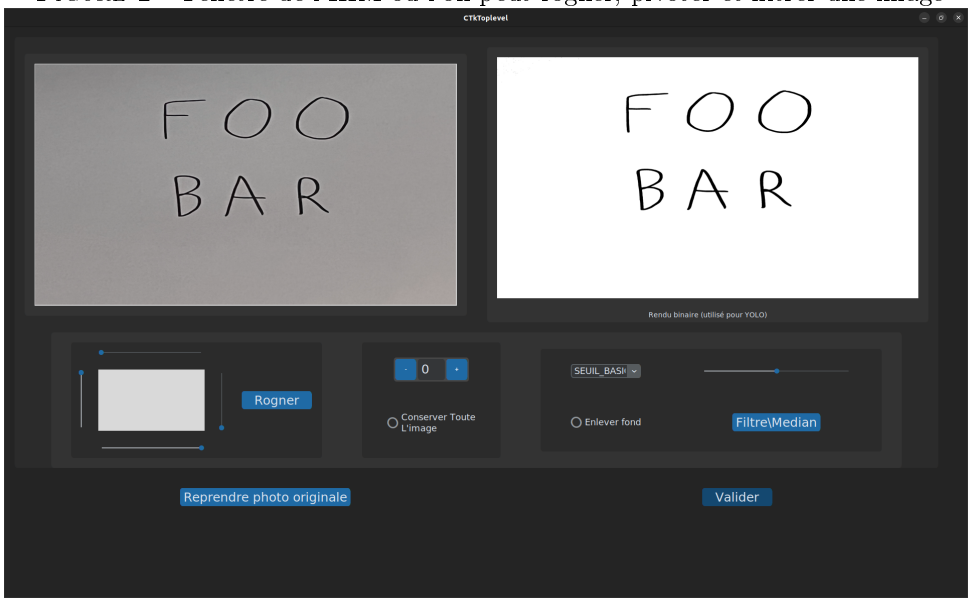
FIGURE 1 – Diagramme de cas d'utilisation



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, il est possible qu'il y ait beaucoup de bruit, dans ce cas là, nous pouvons rogner l'image pour enlever des ombres sur les bords, faire des rotations et filtrer pour enlever le bruit comme illustrée sur la figure 2

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

Sur la figure 3, on peut voir qu'après avoir obtenu une image de bonne qualité, notre programme va segmenter les charactères, en faire des imagettes et donner ces imagettes individuellement à YOLO. Ensuite après cette reconnaissance, l'IHM présente le résultat.

Il est possible de voir les imagettes générées dans le dossier "./images/imageSegmentee"

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

$ndgImage \leftarrow convertirImageEnNDG(image)$;

/* 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 colomnes 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 un caractère grâce aux coordonnées calculées */
/* et on l'ajoute à notre tableau d'image des 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 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
  $redimensionnerImage(imagesCaracteres[index], hauteur = 128, longeur = 128)$;
end
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 9).

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 commposante 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

```
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]pouride0taille(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 charactères crée à partir de la 1e ligne

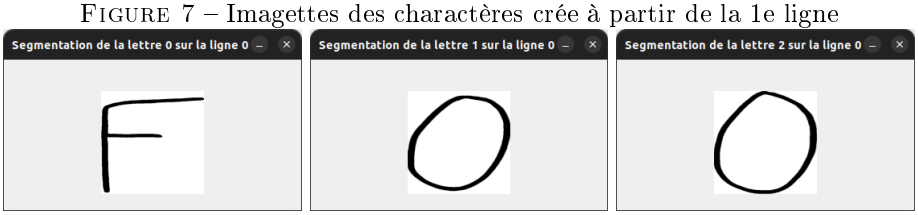


FIGURE 8 – Imagettes des charactères crée à partir de la 2e ligne

Maintenant que nous avons isoler les lignes, nous allons maintenant découper les lettres de chaque lignes en suivant l'algorithme n°2 page 11

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°2 va parcourir totalement l'histogramme de projection et va tenté de dectecter ces "zones pleines" 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).

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 –

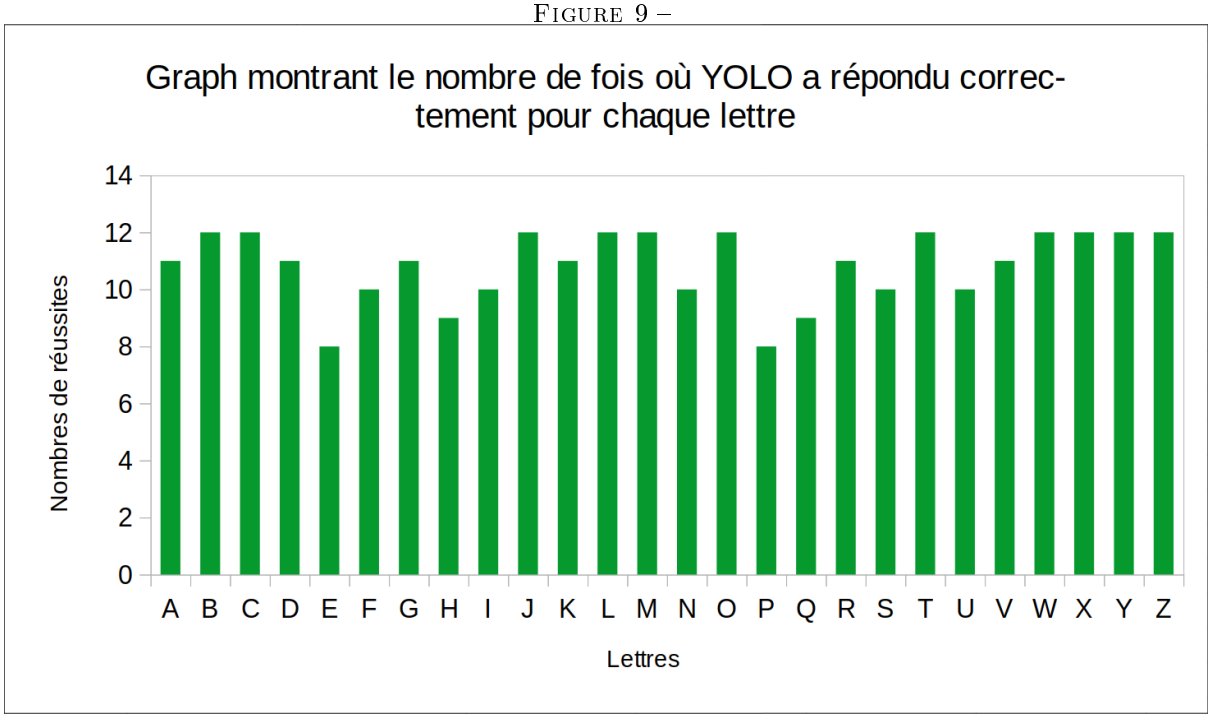
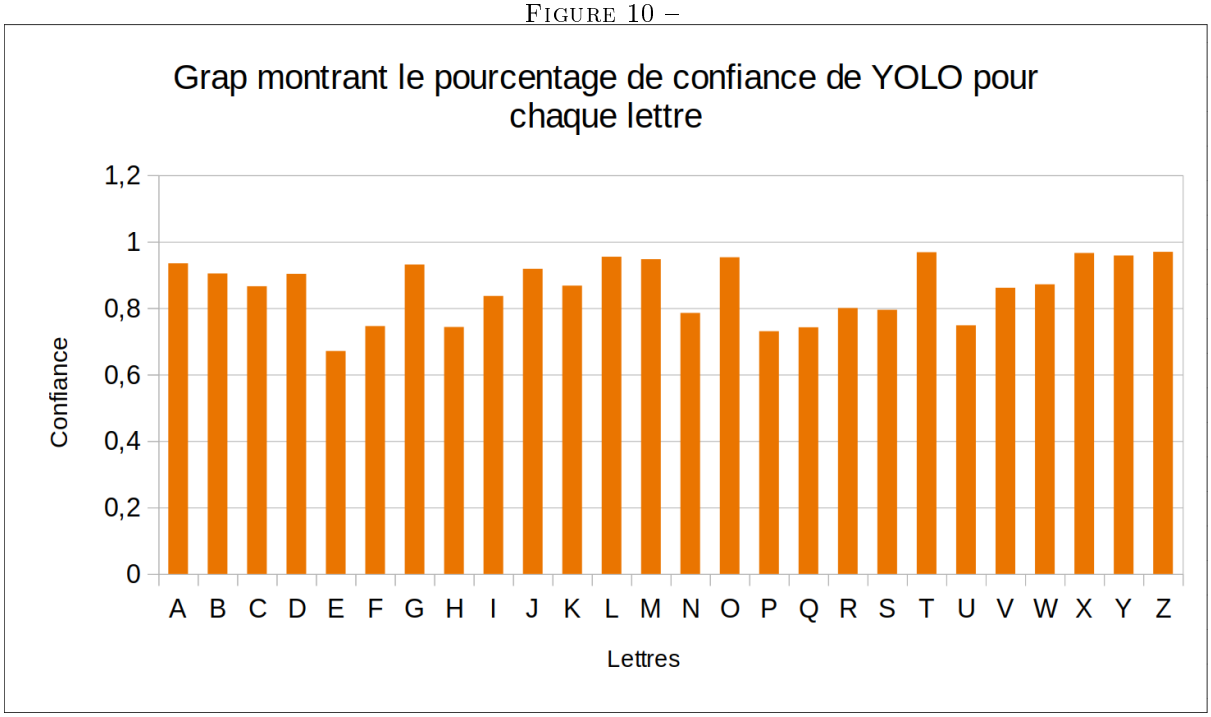


FIGURE 10 –



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 charactères, 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

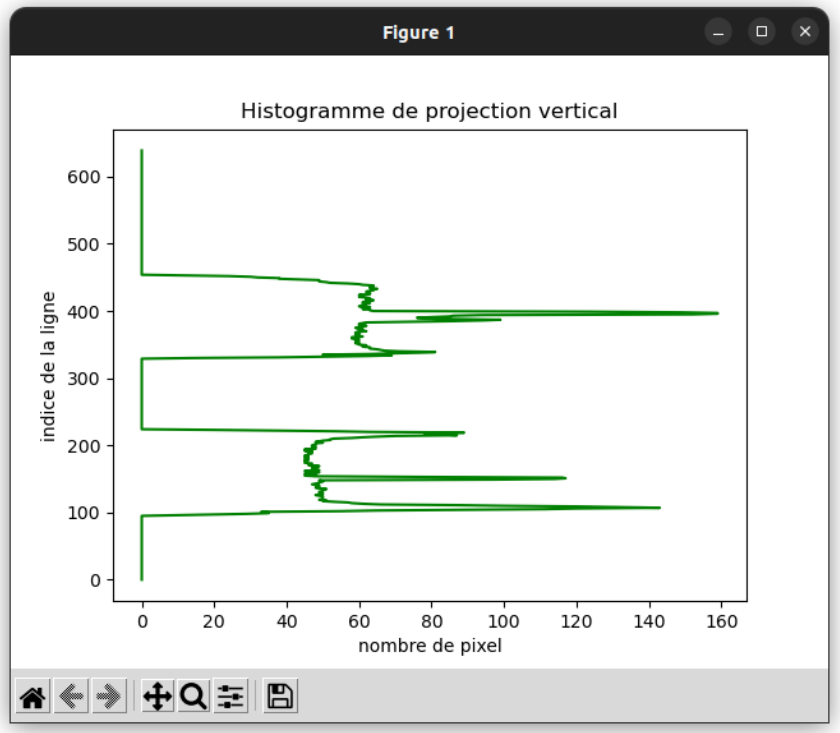


FIGURE 12 – Histogramme de projection horizontal

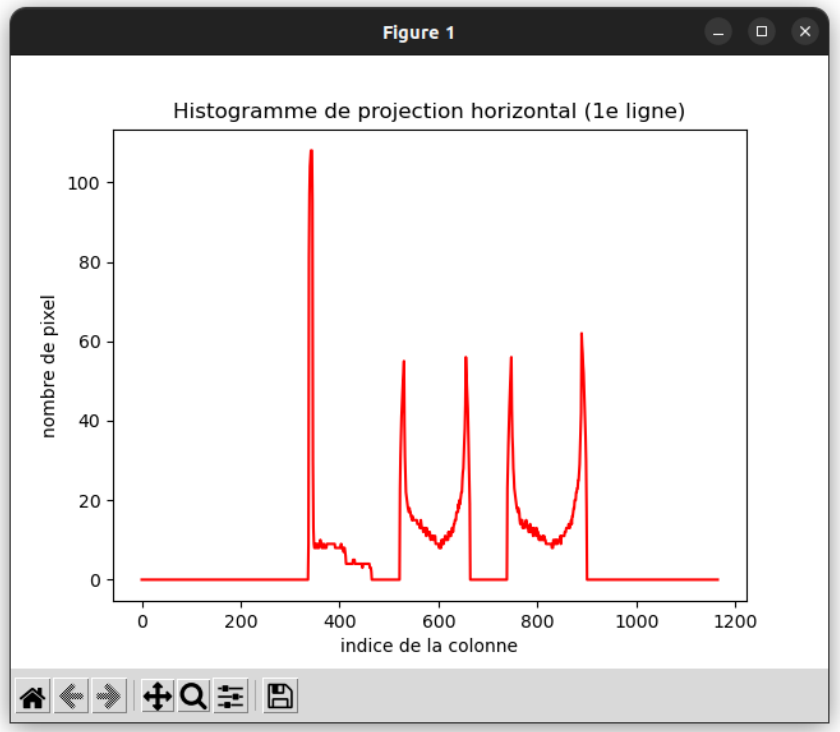


FIGURE 13 – Histogramme de projection horizontal