

Cahier de TP 1 : Lecture automatique des chiffres par analyse d'images

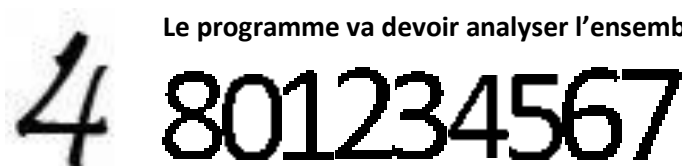
1) L'objectif du TP


L'objectif global est de créer un programme de reconnaissance d'image. Le but final est de pouvoir donner une image avec un chiffre à notre programme et qu'il reconnaisse au mieux ce chiffre via une banque de données composé de chiffres.

Exemples concrets :


Si je donne à mon programme l'image ci-dessous :


Le programme va devoir analyser l'ensemble des photos suivantes :



Dans le cas d'un bon fonctionnement le programme devra sortir : 

Une évolution de ce programme pourrait donner en paramètre une image composée de plusieurs chiffres :



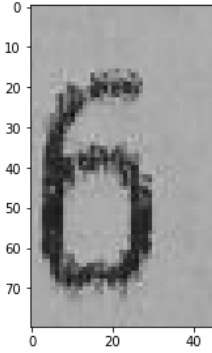
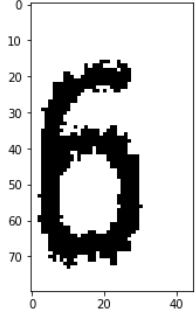
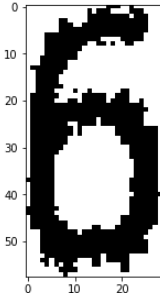
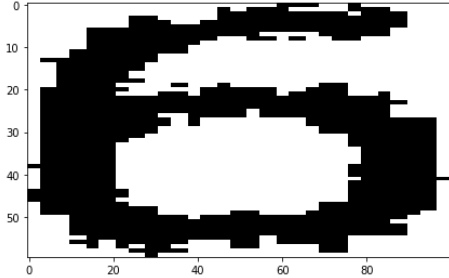
Un retour ce programme pourrait donc être : 

2) Le déroulement du TP

Pour permettre la reconnaissance de ces images, nous avons simplement comparé des images.

Pour la comparaison des traitements, nous avons dû avoir les images du même type pour avoir un taux de réussite maximum. Les étapes ont donc été les suivantes :

Pour l'explication du déroulement du TP nous allons prendre l'exemple des traitements avec l'image « 6 ».

<p>Etape 1 : Récupération de l'image de base</p> 	<p>Etape 2 : Binarisation de l'image (c'est-à-dire rendre l'image en noir et blanc pour faciliter la comparaison)</p> 
<p>Etape 2 : Localisation/Recadrage de la photo (c'est-à-dire recadrer et supprimer les zones blanches inexploitables)</p> 	<p>Etape 4 : Permettre de recadrer l'image pour avoir l'ensemble des images avec une taille équivalente</p> 

Etape 5 : Maintenant que nous avons notre image prête à être comparée, nous devons modifier les images de notre banque de données pour avoir une comparaison facilitée.

Nous effectuons alors pour chaque image de cette banque de données une binarisation, une localisation, et un recadrage en fonction de la taille de l'image de base.

Etape 6 : Dès lors que nous avons des images comparables nous pouvons comparer les images, nous effectuons donc une comparaison de l'ensemble des pixels pour chaque image, et nous ressortons l'image qui semble avoir le plus de ressemblance.

(Etape 7 : Pour aller plus loin) Une évolution de notre programme pourrait permettre la reconnaissance de chiffres venant d'une image composant plusieurs chiffres. Pour cela il suffit de découper les chiffres et de faire le traitement sur chacune des images.

3) Réalisation et résultats

3.1 La binarisation

L'objectif de la binarisation est de rendre l'image plus lisible et plus simple à utiliser. Pour cela on cherche à avoir que deux couleurs présentes : le noir et le blanc.

Pour choisir un niveau de binarisation nous admettons un seuil, ce seuil permettra de choisir à partir de quel moment les pixels passeront du foncé ou au clair.

Pour faciliter cette recherche, et cela pour l'ensemble du TP, un seuil de binarisation à 150 est fixé. Il permet de garder un bon équilibre, en gardant le moins de pixels superflus et évitant la suppression de pixels importants.

Une mise en place d'une fonction pour répondre à ce problème peut être la suivante :

```
def binaris(self,seuil):  
  
    # création d'une image vide  
    im_binarise = image()  
    # affectation de l'image avec la même taille que l'image de base  
    im_binarise.set_pixels(np.zeros((self.H,self.W), dtype=np.uint8))  
  
    # parcours de l'ensemble des pixels en hauteur et en largeur  
    for l in range(self.H):  
        for c in range(self.W):  
            # Verification du seuil passé en paramètre  
            if self.pixels[l][c] >= seuil:  
                im_binarise.pixels[l][c] = 255  
            else :  
                im_binarise.pixels[l][c] = 0  
    return im_binarise
```

Dans cette fonction nous parcourons l'ensemble des pixels de l'image, nous vérifions le code couleur de chacun d'entre eux, et en fonction du seuil et de leur valeur, il passe au noir ou au blanc.

Résultat :

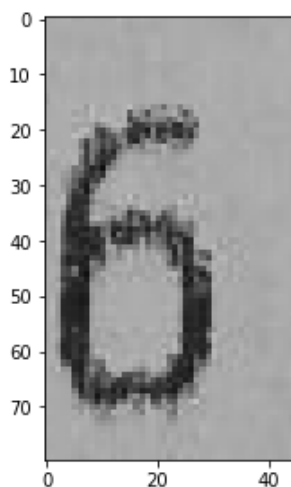


Image de base sans aucune modification

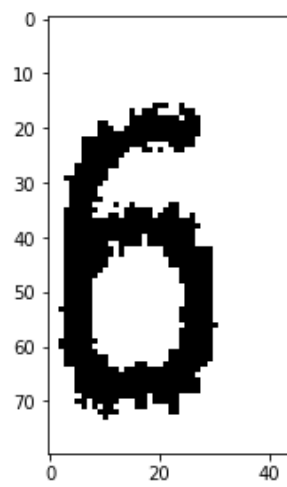
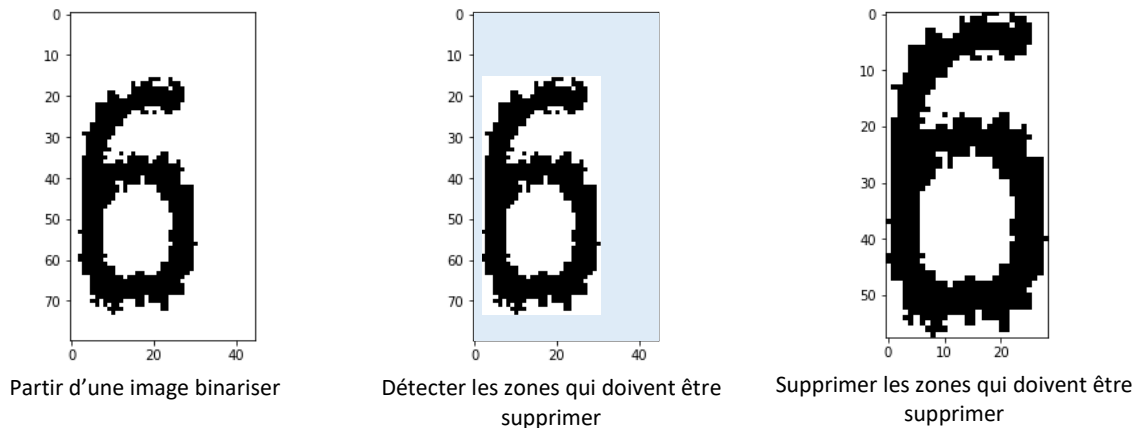


Image après la binarisation à un seuil de 150

3.2 La localisation (le recadrage)

L'objectif de la localisation est d'avoir une image en sortie qui permet une étude uniquement portée sur l'objet en question. La fonction supprime l'ensemble des zones qui ne correspondent pas à l'objet d'étude (dans notre cas des zones blanches).

L'objectif de ce traitement est le suivant, illustré via l'exemple :



La détection des zones qui doivent être supprimés peut se faire en parcourant l'ensemble des lignes et colonnes du tableau. Pour chacune de ces informations, si nous avons un pixel noir cela veut dire que nous devons garder cette ligne, et inversement si la ligne est blanche.

Ensuite, nous pouvons pour trouver les extremums, nous récupérons le numéro de première, dernière, plus à gauche et plus à droite ligne ou colonnes.

3.2 La fonction de redimensionnement

Pour redimensionner une image, nous pouvons utiliser notre fonction `resize_im()`

```
def resize_im(self, new_H, new_W):  
    # Création d'une nouvelle image vide  
    im_resized = image()  
  
    #Création d'un tableau de pixels en fonction des nouveaux H et W  
    tableau_de_pixels = resize(self.pixels, (new_H, new_W), 0)  
    tableau_de_pixels = np.uint8(tab_pixels*255)  
  
    #Ajout du tableau de pixels dans l'image  
    im_resized.set_pixels(tableau_de_pixels)  
  
    return im_resized
```

En prenant en paramètre une image ainsi qu'une nouvelle valeur de largeur et de hauteur, le programme nous envoie une nouvelle image redimensionner. Cette fonction est utile pour permettre la comparaison entre deux images de tailles différentes. Comme nous le verrons plus tard, souvent nous modifier la taille de l'image du panel en fonction de l'image à tester.

La fonction est utilisable sur tout type d'image (binarisé, non binarisé, recadré ou non) :

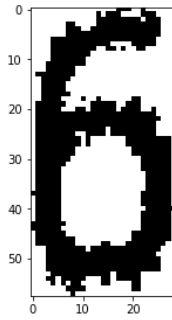


Image de base (60x30)

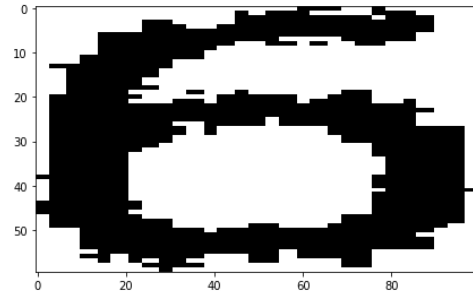


Image après passage dans la fonction resize avec comme nouvelle valeur (60,100)

3.3 La fonction similitude

La fonction similitude permet la comparaison entre deux images, avant de faire une comparaison entre deux images nous devons faire l'adaptation de ces images pour avoir un environnement de comparaison cohérent.

Dans ce cas les images de notre banque de données sont déjà binarisé et recadré, mais il aurait été important de le faire si ce n'était pas le cas. Notre image qui nécessite une comparaison doit elle être binarisé, puis recadré (localiser). Nous pouvons ensuite recadrer l'image en fonction de l'image de la banque de test.

Ensuite lorsque nous avons deux images cohérentes nous pouvons parcourir l'ensemble des pixels des deux images pour les comparer, nous aurons donc un pourcentage de ressemblance. Sur l'ensemble des comparaisons nous pouvons sortir l'image qui a le pourcentage de ressemblance le plus important.

Exemple : Si nous donnons à notre programme l'image suivante :



Le programme va alors comparer cette image à l'ensemble de notre banque de données :

8901234567

Nous réalisons ensuite les différents traitements depuis l'image à tester sur les images tests, et notre programme nous retourne alors une image avec un taux de ressemblance.

Ici 94.14% avec l'image suivante :



Cette fonction est également fonctionnelle avec l'ensemble des images de la banque de test mais peut poser des problèmes sur la comparaison de différents chiffres. En effet nous avons tous une écriture différente, si une personne donne au programme un chiffre déformé, avec un style d'écriture radicalement différent le programme pourra échouer sa reconnaissance.

Il sera alors intéressant de faire une interprétation sur un grand nombre de chiffre pour ensuite pouvoir tester et répondre à l'ensemble des cas possible. Plus le nombre de tests pour la détection d'un même chiffre est élevé, plus le taux de réussite sera important.

4) Problèmes rencontrés lors du TP

Durant ce TP j'ai dû rencontrer un problème de taille, en effet lors du début de ma phase de développement, mes programmes me retournaient des valeurs parfois incohérentes.

Ces incohérences venaient uniquement dans certains cas, j'ai dans un premier temps remis en cause le fonctionnement de mon programme, en réécrivant l'ensemble de mes fonctions, et je me suis entêté à répondre au problème par moi-même.

Après avoir passé des longues heures à chercher une solution j'ai demandé à un camarade de tester mon propre code sur son ordinateur avec les mêmes fichiers de tests. Les résultats étaient finalement corrects et validaient la correspondance des images.

Le problème venait donc pas de moi, mais j'ai voulu comprendre ce problème .. Spyder ne me renvoyait aucune erreur, aucun problème apparent.

Après de longues recherches sur des forums sur internet j'ai découvert mon problème :

"UserWarning: Anti-aliasing will be enabled by default in skimage 0.15 to avoid aliasing artifacts when down-sampling images. warn("Anti-aliasing will be enabled by default in skimage [...])"

Le problème venait alors d'une mauvaise mise à jour du module python scikit-image, problème qui s'est ensuite supprimé grâce à cela.

Ce problème et ses conséquences m'ont fait perdre du temps, rongé le temps pour finir proprement ce TP ainsi que la partie « pour les plus rapide ».