# Rapport de TP2 – Lecture automatique de chiffres par analyse d’image

## Introduction

Lors de cette deuxième séance de TP, nous allons pouvoir mettre en pratique les notions abordées dans le cours afin de mettre au point un programme de reconnaissance automatique de caractères (ici des chiffres) en analysant une image (en niveaux de gris) chargée dans le programme. Comme expliqué dans l’énoncé du TP, cette démarche s’opérera en 5 étapes majeures :

* La binarisation va permettre de transformer l’image en niveaux de gris en une image avec des pixels soit noirs, soit blancs.
* La localisation sera utile à restreindre l’image afin de n’englober que la partie contenant le chiffre à déterminer.
* L’adaptation de la taille aux modèles adaptera la taille de l’image contenant le chiffre à identifier à la taille des modèles préchargés.
* La mesure de ressemblance par corrélation va mesurer la ressemblance d’une image aux modèles. Le résultat sera un nombre compris entre 0 et 1.
* Une décision sera prise afin de choisir le chiffre qui a le plus de corrélation avec l’image.

## Travail préparatoire

Dans cette partie, nous allons comprendre le fonctionnement de la classe image et créer les méthodes de binarisation et de localisation, qui seront utiles pour la reconnaissance automatique de chiffres.

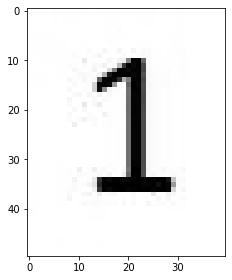
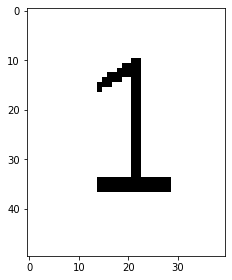
### Analyse de la classe image

En observant de plus près la classe image, nous comprenons que celle-ci permet de créer des attributs H,W et pixels tels que H et W sont les dimensions de l’image (H = high = hauteur et W = width = largeur). Ces dimensions sont donc le nombre de pixels en hauteur et en largeur. L’attribut pixel est un tableau qui contient une valeur comprise entre 0 et 255 indiquant l’intensité de la couleur (avec 0 = noir et 255 = blanc).

### Écriture de la méthode binarisation(self, S)

Cette méthode va nous permettre de traiter l’image en la passant d’une image composée de niveaux de gris en une image composée de noir et de blanc, en puissant sélectionner l’intensité de détection des pixels. Pour cela, nous utilisons le tableau des pixels de l’image et créons un autre tableau de mêmes dimensions, rempli de 0. Avec deux boucles for imbriquées, nous traitons l’ensemble des pixels et changeons la valeur du nouveau tableau à 255 quand la valeur de l’ancien tableau dépasse le seuil S sélectionné par l’utilisateur.

Ce code nous permet d’obtenir une image en noir et blanc à partir de celle en niveaux de gris. En l’utilisant avec l’image donnée et un seuil donné à 70, on a le résultat suivant :

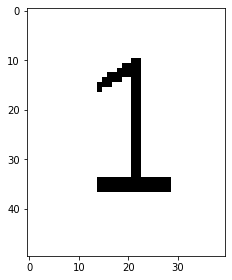


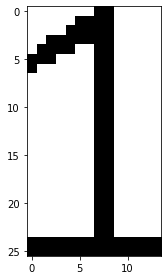
On observe bien une image résultante binarisée, qui ressemble toujours au 1 de l’image initiale. Il n’y a pas d’erreur dans le codage des boucles imbriquées car l’image est à l’endroit.

### Écriture de la méthode localisation(self)

Cette méthode-ci va permettre de recadrer l’image afin d’avoir une image de dimensions minimales qui contient le chiffre à déterminer. Pour cela, nous avons effectué 4 boucles while qui vont déterminer les limites du chiffre (haut, bas, gauche et droite) afin que tous les pixels noirs du chiffre soient à l’écran.

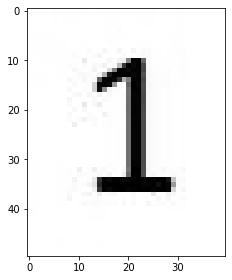
Ce code nous permet d’obtenir le résultat suivant :

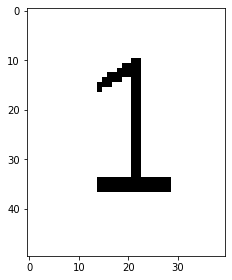


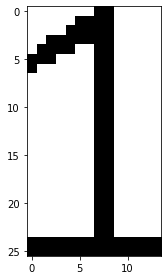


Le résultat semble correct car le chiffre n’est pas coupé, il ne manque pas de pixel noir et le chiffre est à l’endroit.

La préparation de ce TP nous aura donc permis d’écrire 2 méthodes nous permettant de binariser l’image et de localiser le chiffre sur l’image en question, tout cela en 2 étapes, selon le schémas suivant :







Localisation

Binarisation

## Reconnaissance automatique de chiffre

### Contrôle du bon fonctionnement de la méthode binarisation

Afin de vérifier qu’il n’y a pas de problème de fonctionnement de la méthode susmentionnée, nous lançons le programme test\_image.py. Les seuls échecs et erreurs affichés ne concernent que les méthodes qui seront à programmer par la suite, qui apparaissent donc comme des erreurs ou échecs car elles ne sont pas encore codées.

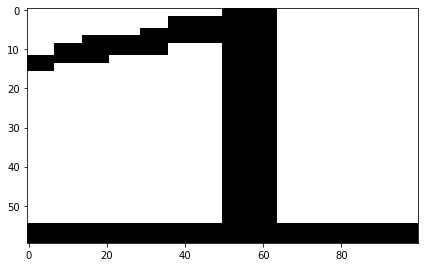
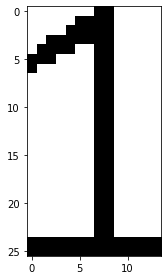
### Contrôle du bon fonctionnement de la méthode localisation

De même que pour la partie précédente, nous lançons le programme test\_image.py et toujours identiquement à la méthode de binarisation, nous n’observons pas d’erreurs ou échecs concernant cette méthode.

### Écriture de la méthode resize(self,new\_H,new\_W)

Dans cette partie, nous allons chercher à programmer une méthode permettant de redimensionner une image avec des dimensions voulues. Pour cela, on utilisera la fonction resize de la librairie skimage.

En entrant les dimensions (60,100), on obtient bien une image dont les dimensions ont été modifiées aux valeurs données :



Une fois le programme test\_image.py lancé, on voit bien qu’il n’y a pas de problème concernant cette partie de programmation.

### Écriture de la méthode similitude(self, image)

Cette méthode a pour objectif de mesurer un taux de correspondance entre 2 images de même dimensions, qui sera compris entre 0 et 1. Nous allons donc créer deux boucles for imbriquées afin de parcourir la totalité des images et d’incrémenter un compteur à chaque fois que les pixels sont identiques. A la fin de la boucle, on effectue le rapport du nombre de pixels identiques sur le nombre de pixels au total.

Lorsque nous lançons le programme de test, aucune erreur n’est détectée. Cela veut donc dire que le programme renvoie bien un nombre à virgule compris entre 0 et 1 et qui est bien proportionnel à la corrélation entre deux images.



Tout ce qui a été effectué auparavant est donc valide car aucun échec et aucune erreur ne sont affichés.

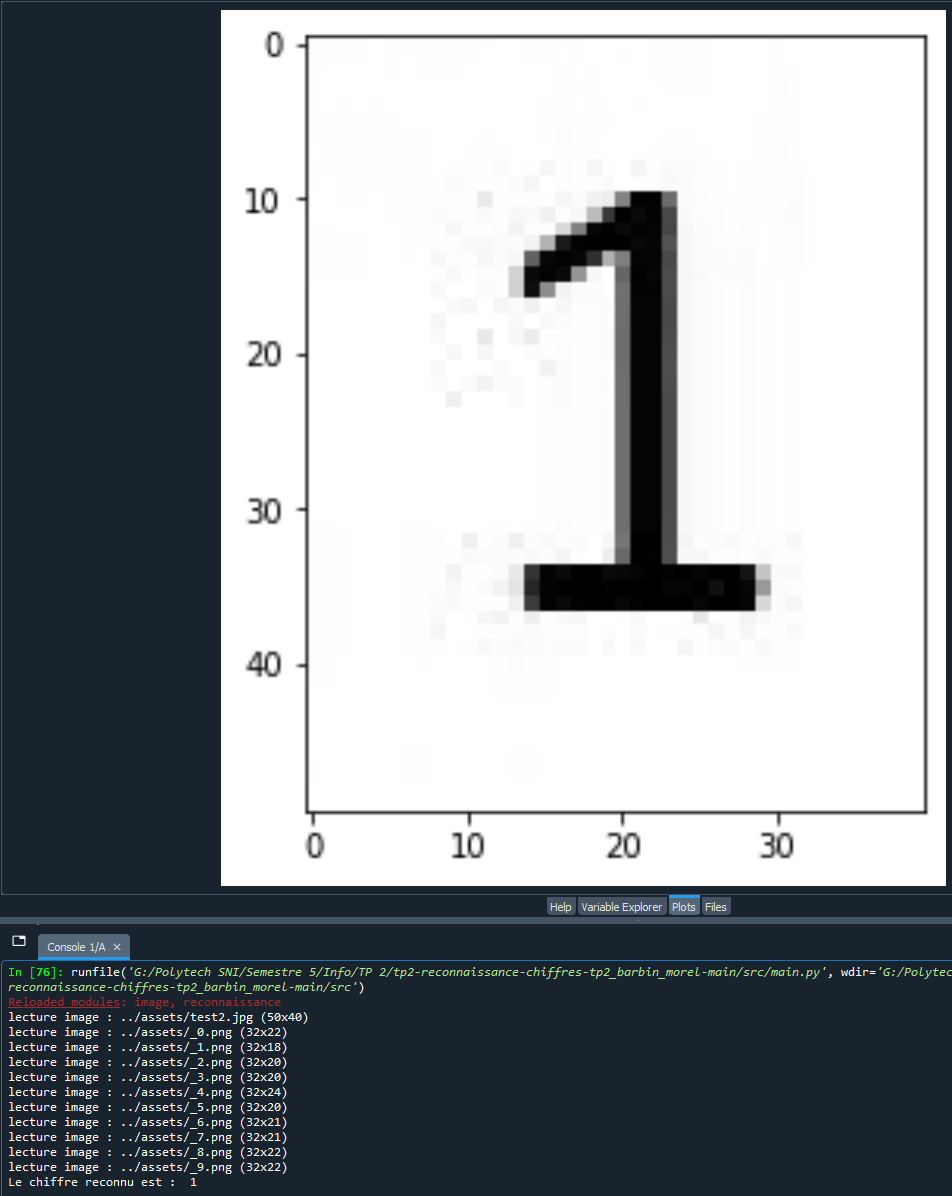
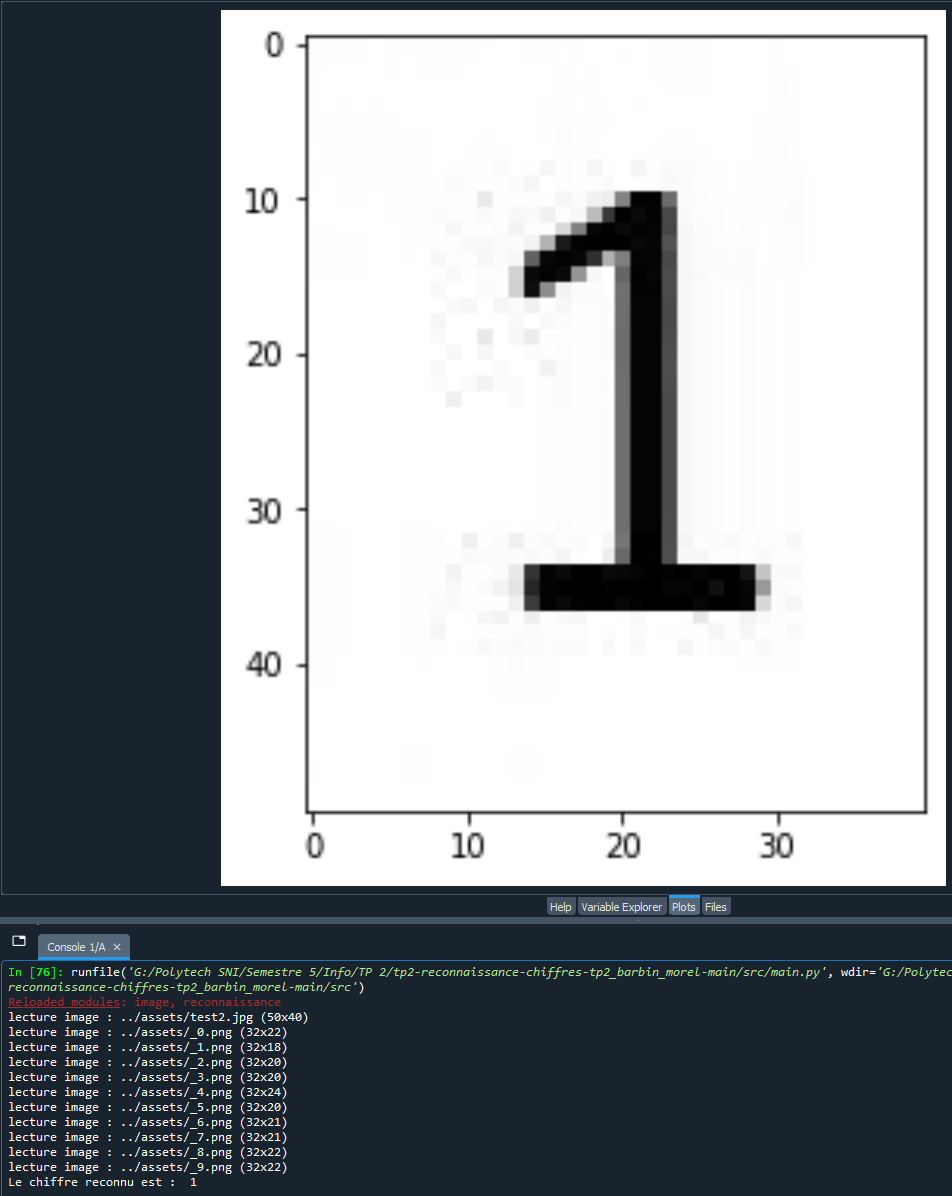
### Écriture de la fonction reconnaissance\_chiffre(image, liste\_modeles, S)

Cette dernière fonction à créer va permettre de déterminer le chiffre écrit sur l’image entrée en plusieurs étapes. Tout d’abord la fonction va binariser l’image en utilisant la méthode adéquate, puis la localiser. Ensuite pour chaque modèle disponible, la fonction va mettre l’image localisée dans les mêmes dimensions que le modèle et calculer la similitude avec ce modèle. A chaque tour de boucle, si la similitude correspondant au modèle est supérieure à la dernière similitude retenue, le programme mémorise le chiffre correspondant au modèle (le numéro du tour i) et met à jour la valeur de la similitude pour les tours de boucle suivants.

Lorsque nous lançons le programme de test, aucune erreur n’est détectée. Cela veut donc dire que le programme détecte le bon chiffre.



En lançant le main, le programme renvoie bien le chiffre correspondant à l’image :



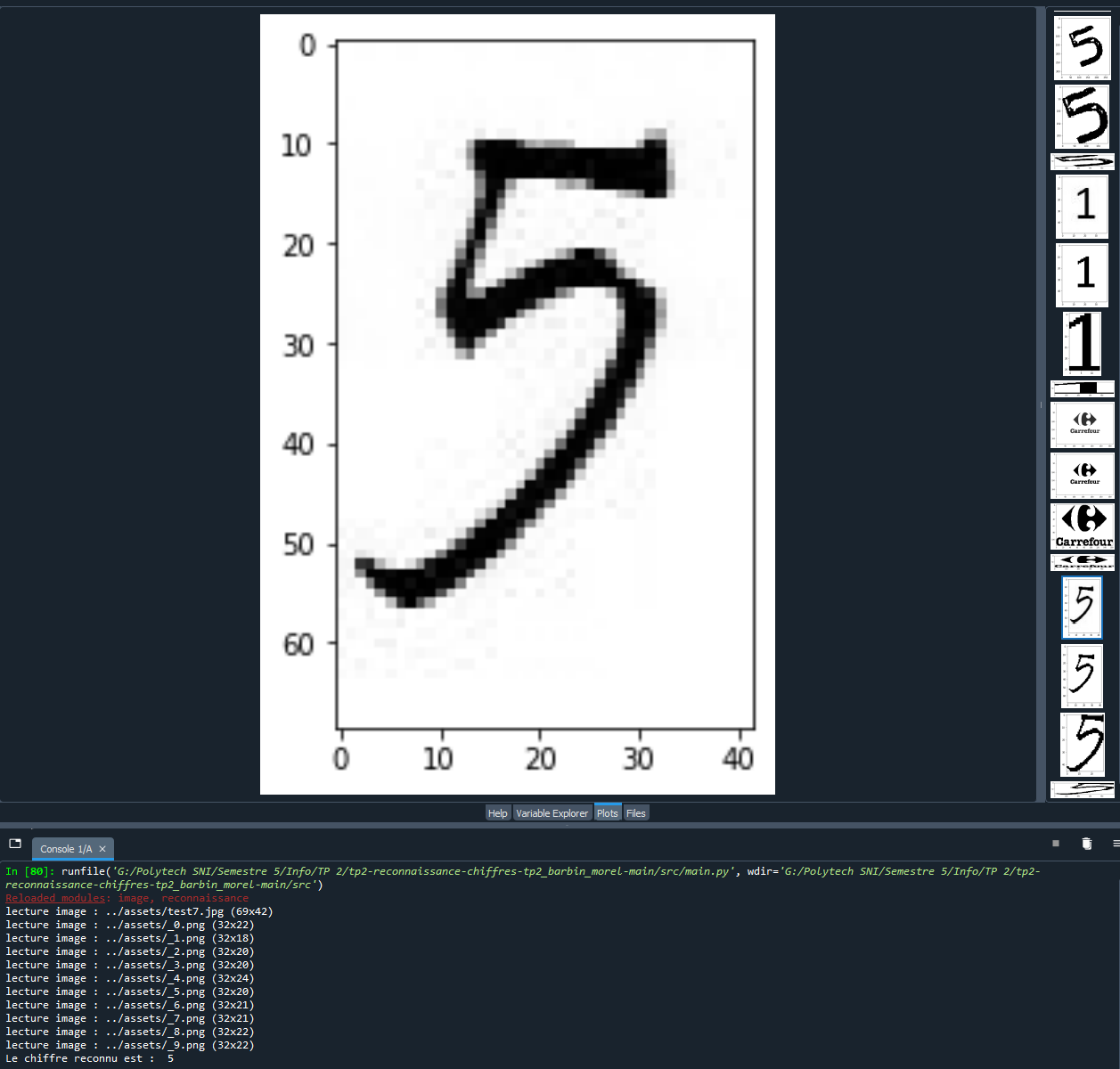
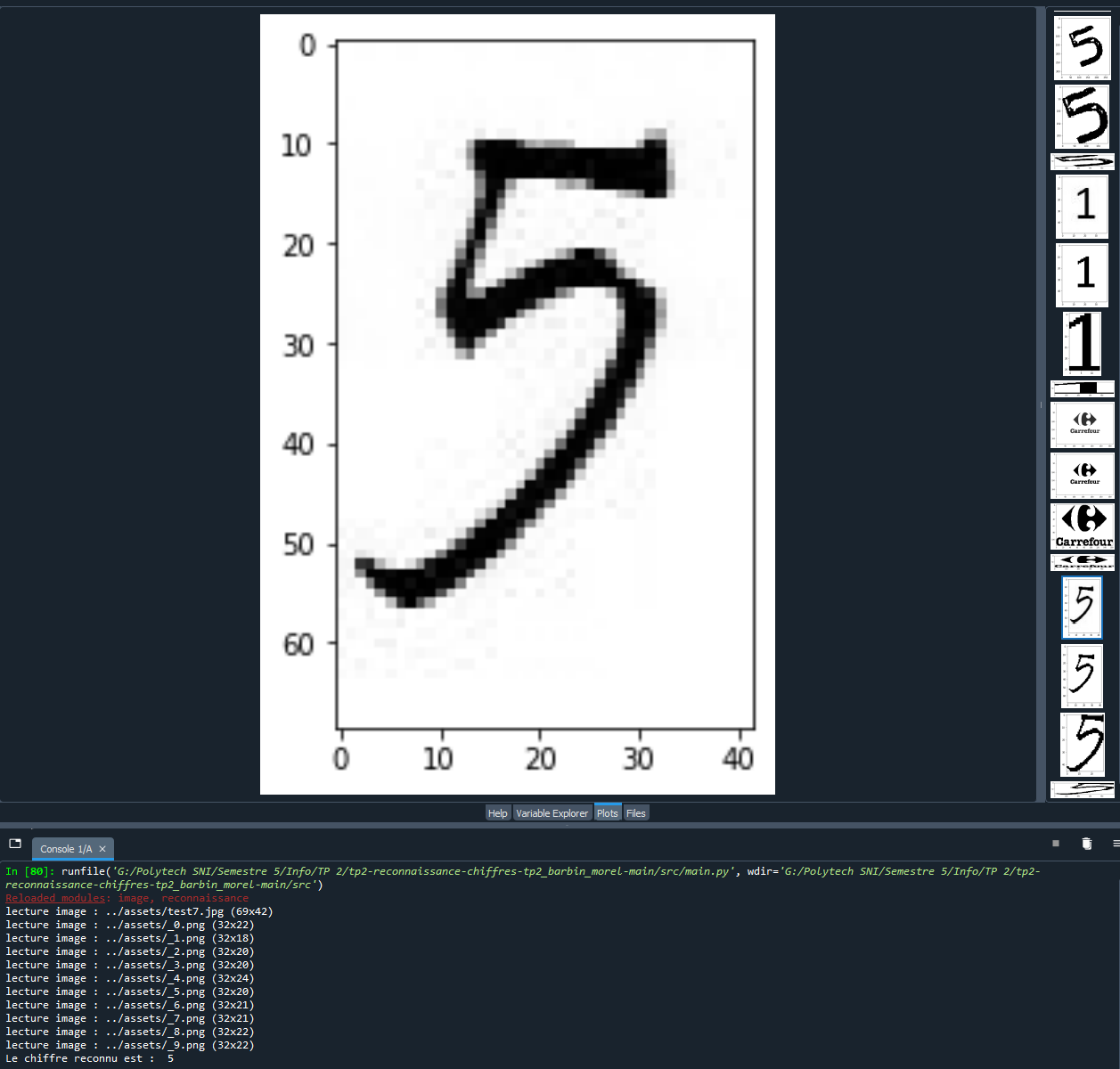
Resize

Localisation

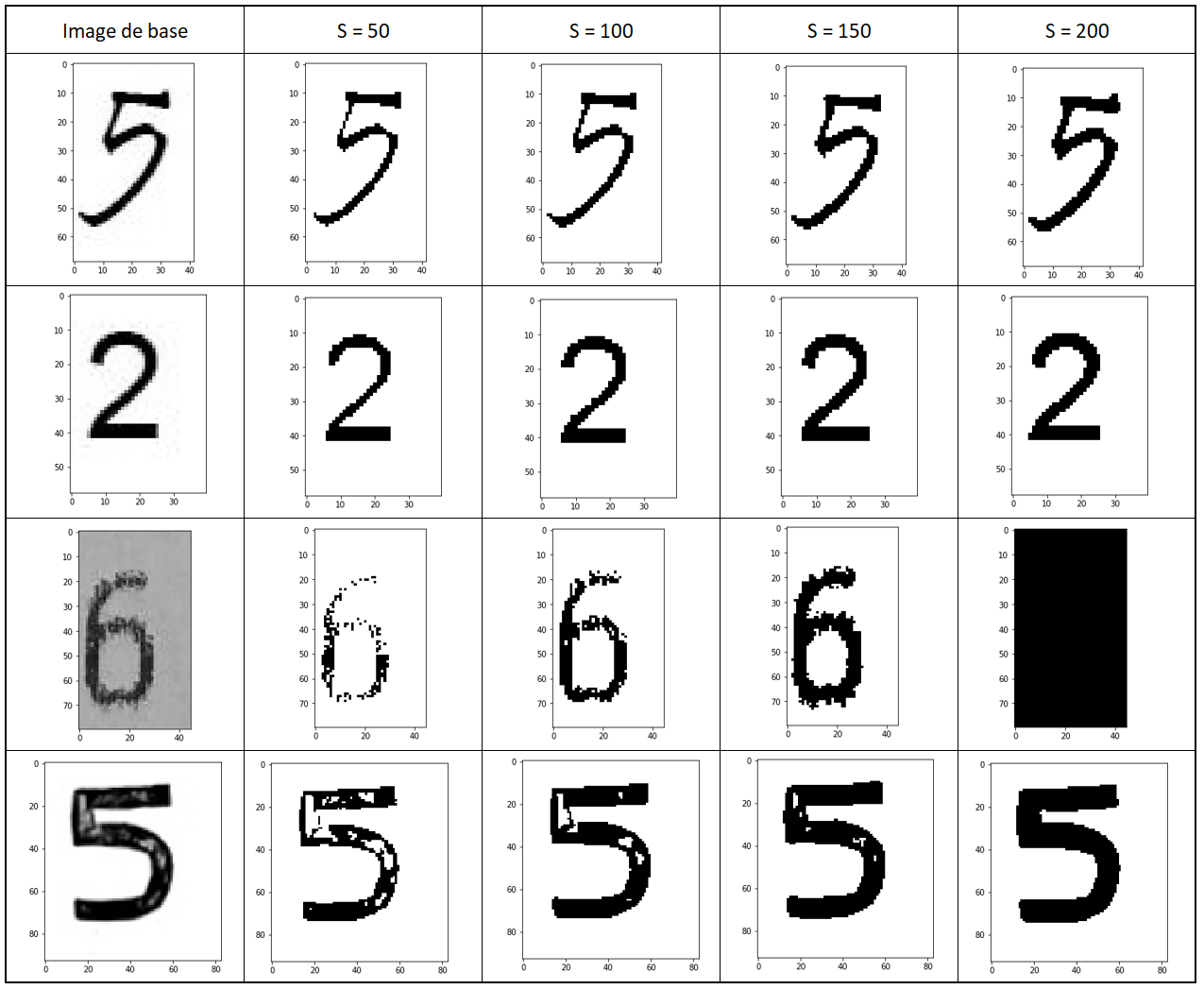
Binarisation

### Test avec une autre image et détermination du seuil optimal

On remplace l’image à analyser par le chiffre 5 (image nommée test7) et on obtient le résultat suivant avec un seuil de 70 :



Afin de déterminer le seuil optimal à la détection du chiffre, on réalise plusieurs essais répertoriés dans le tableau suivant :



(\*)

(\*) Image dessinée par nos soins sur papier et prise en photo avec un filtre noir et blanc.

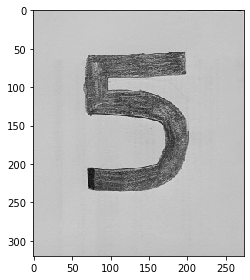
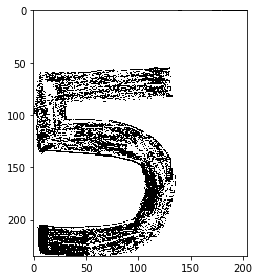
Plus on augmente le seuil de détection, plus l’image contient de pixels noir. Cela peut poser des problèmes, car avec un seuil trop élevé, les défauts des images apparaîtront (si c’est une photo : des poussières ou encore le grain du papier). Si le seuil est trop faible, le chiffre n’est pas totalement détecté. On propose d’utiliser un seuil compris entre 100 et 150. On choisira un seuil de 100.

## Pour aller plus loin

### Problème de localisation lié aux défauts

Nous allons tester le programme avec des images que nous avons créées. Pour que l’image puisse être traitée par le programme, nous avons noté que celle-ci doit être en niveaux de gris. Nous avons également remarqué que s’il y avait des défauts en périphérie de l’image, la localisation ne s’effectue pas correctement.

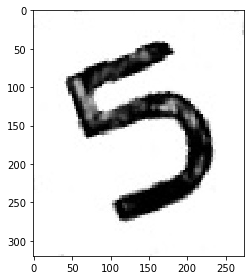
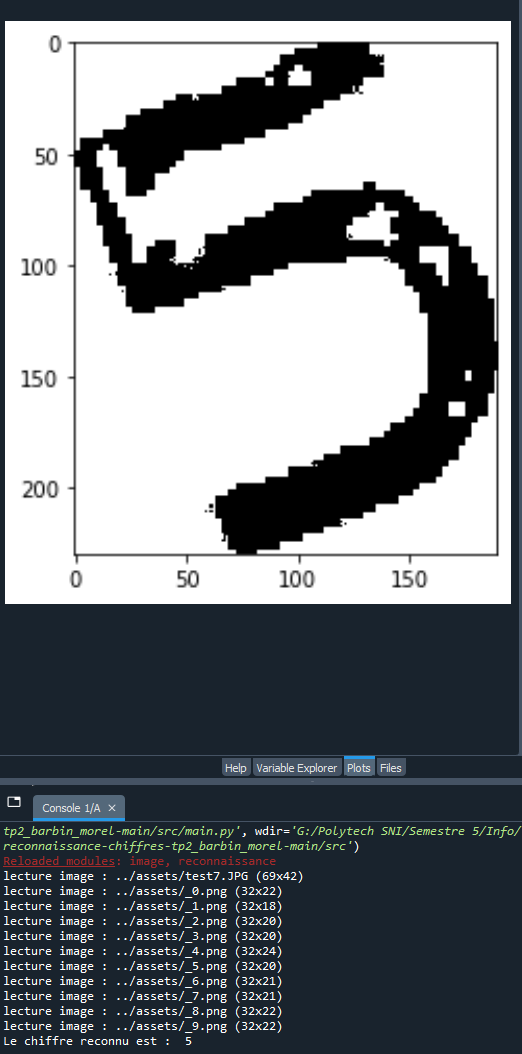
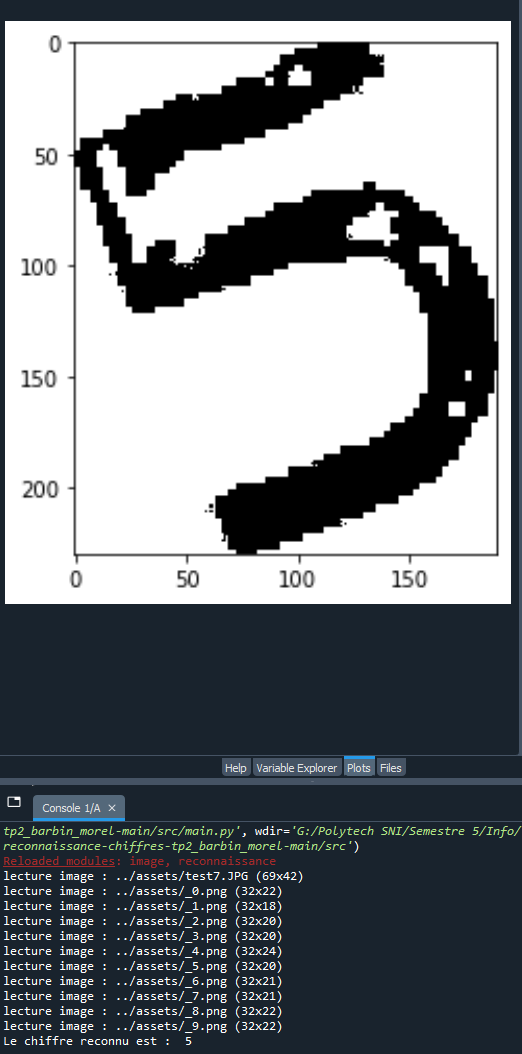
En voici un exemple avec un chiffre que nous avons dessiné :

La présence de certains pixels noirs autour du chiffre empêche de localiser le chiffre correctement.

### Orientation de l’image

Nous avons aussi voulu tester si le programme détecte les chiffres malgré une orientation de 20° de l’image :

## Conclusion

Durant ce TP, nous avons pu créer différentes fonctions au sein d’une classe dans le but de retrouver le chiffre de l’image. Pour cela, nous sommes passés par différentes étapes, telles que la binarisation qui transforme les couleurs de l’image qui sont sous la forme de nuances de gris en deux valeurs (noir et blanc), la prochaine étape est la localisation qui recadre l’image autour du chiffre. Enfin, pour chacun des modèles, l’image est redimensionnée selon le modèle pour que le programme puisse les comparer et nous afficher le chiffre qui lui semble être le plus ressemblant. Ces différentes fonctions nous auront permis de revoir différentes structures (for, if), l’utilisation de listes et de classes abordés en cours et mis en situation dans une application concrète. Enfin, nous avons voulu créer notre propre image avec un chiffre que nous avons dessiné et l’appliquer à notre programme, ce qui a fonctionné et nous a poussés à faire des tests de plus en plus exigeants.