Informatique

Intelligence artificielle

Réseau de neurones et knn
Reconnaissance d'écriture

Présentation du sujet

Nous souhaitons utiliser des algorithmes de l'intelligence artificielle appliqués à la reconnaissance d'écriture en caractères non attachés (on se limitera aux chiffres de 0 à 9 manuscrits).

Nous souhaitons ainsi que l'ordinateur puisse partir de l'image de gauche ci-dessous, et donner le résultat à droite automatiquement :

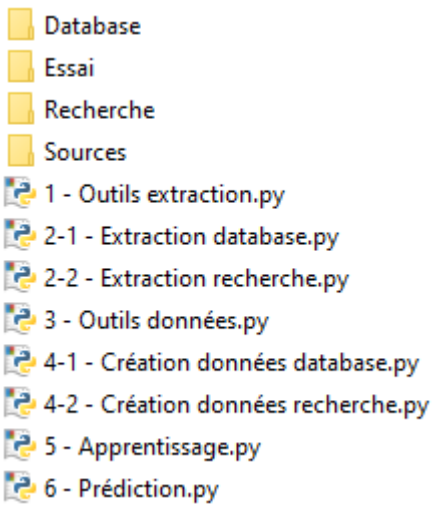
La seule condition à respecter pour ce travail est d'avoir des caractères en un seul « paquet » de pixels. Nous supprimeront automatiquement des petits paquets isolés de quelques pixels, mais il ne doit pas y avoir plusieurs « grosses » zones pour un seul caractère.

Au programme, nous devons aborder les réseaux de neurones (RN), la méthode des k plus proches voisins (KNN) et la régression linéaire multiple (RLM) (cette dernière méthode ne sera pas appliquée ici puisque nos données ne présenteront pas de caractère linéaire).

Nous allons dans le cadre de différents exercices, réaliser les choses suivantes :

- Exercice 1 : Développement d'outils d'extraction de caractères dans une image
- Exercice 2 : Extraction des images de la database et de l'image recherchée
- Exercice 3 : Développement d'outils de création des données numériques des images extraites
- Exercice 4 : Création des données numériques associées aux images extraites
- Exercice 5 : Mise en place de l'apprentissage des algorithmes d'IA
- Exercice 6 : Reconnaissance des images extraites de l'image recherchée

Il est possible de commencer ce sujet à partir de la partie 5 (uniquement la partie IA), vous travaillerez alors dans le « Dossier élèves 5 à 6 » (les codes 1-1, 2-1, 2-2,3,4-1 et 4-2 seront alors préremplis et les images à disposition dans les différents dossiers). Si vous souhaitez réaliser les parties 1 à 5 pour créer la base de données des images sous Python, procurez-vous le « Dossier élèves 1 à 6 ».
Vous aurez les images sources, les images recherchées, et un code Python à compléter.
Chacun des 6 exercices devra être réalisé dans le/les fichier/s du même nom. Ouvrez donc, dans l'ordre, tous les fichiers sur votre logiciel. Lorsqu'ils seront complétés, il suffira alors de les exécuter un par un dans l'ordre.



Exercice 1: Outils d'extraction

Question 1: Créer une fonction Affiche(fig,im) qui permet d'afficher l'image im (array) sur la figure fig
Question 2: Créer une fonction Lecture_im(Chemin) ouvrant l'image de chemin « Chemin », la transformant en RGB si RGBA et la renvoyant
Question 3: Ecrire les lignes de code créant l'array « Image » associé à « essai.bmp » du dossier « essai » et l'affichant

Dans la suite, l'argument im lors de la définition des fonctions sera cette image en couleurs.

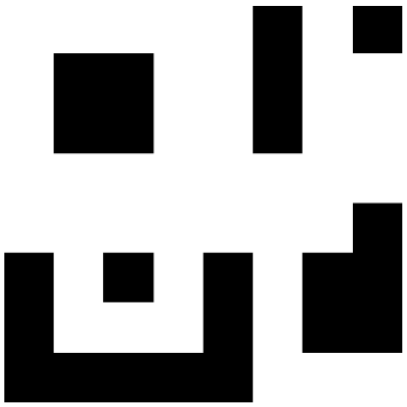
Question 4: Créer une fonction NG(im,kR,kG,kB) qui renvoie une nouvelle image imng (array) qui sera la transformation de l'image im en nuances de gris avec kR, kG et kB les facteurs sélectionnant les couleurs R, G et B de l'image
Question 5: Ecrire les lignes de code créant l'image « Image_NG » pour kR=kG=kB=1/3 et l'affichant

Vérifier :

Dans la suite, l'argument imng lors de la définition des fonctions sera cette image « Image_NG ».

Question 6: Créer une fonction NB(imng,k) qui renvoie une nouvelle image en noir et blanc comme souhaité à partir de l'image imng
Question 7: Ecrire les lignes de code créant l'image « Image_NB » pour k=200 et l'affichant

Vérifier :



Dans la suite, l'argument imnb lors de la définition des fonctions sera cette image.

L'image obtenue ci-dessus présente une grande « zone » blanche et un certain nombre de « zones » noires. Nous allons programmer une méthode permettant d'associer un numéro à chacune de ces zones afin de pouvoir les dénombrer et d'extraire chacune d'entre elle.

Pour chaque pixel de l'image étudiée désigné par une liste [l,c] de sa ligne l et sa colonne c, on souhaite déterminer chacun de ses 4 voisins (gauche, droite, bas, haut) dans cet ordre sous la forme d'une liste de 4 listes [li,ci]. Un voisin qui sortirait de l'image sera défini comme égal au pixel [l,c].

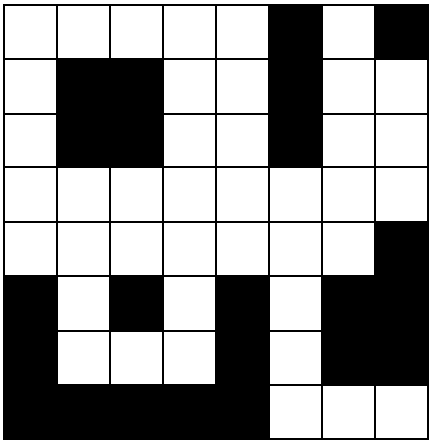
Question 8: Proposer une fonction Liste_voisins(l,c,Nl,Nc) prenant en argument la ligne l et colonne c du pixel étudié, les nombres de lignes Nl et colonnes Nc de l'image, et renvoyant la liste de ses 4 voisins comme précisé ci-dessus

```
>>> Liste_voisins(0,0,10,10)
[[0, 0], [0, 1], [1, 0], [0, 0]]

>>> Liste_voisins(1,1,10,10)
[[1, 0], [1, 2], [2, 1], [0, 1]]

>>> Liste_voisins(9,9,10,10)
[[9, 8], [9, 9], [9, 9], [8, 9]]
```

On suppose avoir à disposition une table T sous forme d'array à deux dimensions, de mêmes dimensions que l'image étudiée, contenant des entiers codés sur 64 bits. Elle est initialisée à des valeurs de -1 partout. On souhaite que cette table contienne par la suite des entiers positifs croissants tels que toutes les cases contenant l'entier k représentent une « zone » numéro k de l'image noir et blanc.

Ainsi, comme le premier pixel en haut à gauche est blanc, la première zone de numéro k=0 sera l'intégralité du blanc de l'image ne formant qu'une zone. La zone 1 sera une première zone noire, la zone deux une seconde, etc.

Voici un exemple de table T associée à l'image de 8x8 pixels ci-contre en réalisant un parcours colonne par colonne (pour chaque colonne, puis pour chaque ligne) de la procédure que nous allons mettre en place :



0	0	0	0	0	4	0	6
0	2	2	0	0	4	0	0
0	2	2	0	0	4	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	5
1	0	3	0	1	0	5	5
1	0	0	0	1	0	5	5
1	1	1	1	1	0	0	0

Ce qui donnera, en affichant la table T avec notre fonction Affiche (lorsque ce sont des entiers, et non des triplets, on obtient un dégradé de couleurs de bleu à jaune automatiquement) :

Dans la suite, on dira qu'une case T[l,c] a été **marquée** si sa valeur T[l,c] est différente de -1.
Par ailleurs, on dira par abus de langage « **pixel [l,c]** » pour désigner le pixel à la ligne l et la colonne c et « **case [l,c]** » la valeur de la case à la ligne l et la colonne c de la table T.
Enfin, comme nous travaillerons sur l'image en noir et blanc, on parlera de « **valeur** » 0 ou 255 pour identifier les valeurs R=G=B de tous les pixels.

On souhaite créer une procédure qui, à partir d'un pixel initial [l,c] quelconque de l'image associé à une case T[l,c] initialisée à un entier k dans T, explore tous ses voisins non encore marqués ayant la même valeur, afin de les marquer du même entier k, et procède ainsi pour tous les voisins des voisins (etc.), tant qu'il y en a. Cette procédure va donc permettre de remplir la table T avec le même entier k pour chaque zone de l'image.

Principe de la fonction d'exploration à partir du pixel initial [l,c] :

- Initialiser une pile (liste) avec le couple [l,c] du pixel initial
- Tant que la pile n'est pas vide :
    - Dépiler un pixel [l,c]
    - Affecter l'entier k à la case [l,c]
    - Déterminer tous les voisins du pixel [l,c]
    - Pour chaque voisin non marqué de même valeur :
        - Ajouter ce voisin à la pile

  Attention : Tout n'est volontairement pas dit

Exemples : L'appel de la fonction en

- [0,0] va créer toute la zone de 0 de la table T
- [5,0] va créer toute la zone de 1 dans la table T

Question 9: Proposer une fonction Explorer(imnb,l,c,T,k) prenant en argument la ligne l et la colonne c du pixel initial et l'entier k, et réalisant la procédure attendue

Il reste maintenant à appeler cette procédure sur l'intégralité des pixels non marqués de la table T sur l'image en noir et blanc à partir du pixel [0,0].

Principe de la fonction de détermination des zones :

- Initialisation d'une table T d'entiers à -1 (`dtype='int64'`)
- Initialisation de l'entier k à 0
- Parcours de tous les pixels colonne par colonne
- Exploration du pixel s'il n'est pas marqué en lui attribuant l'entier k (création de la zone k)
- Incrémentation de l'entier k

  Attention : Tout n'est volontairement pas dit

Question 10: Créer la fonction Zones(imnb) réalisant cette procédure et renvoyant la table T associée à imnb

Question 11: Ecrire les lignes de code créant la table « Table » associée à l'image essai, l'affichant, et créant et affichant dans la console le nombre de zones trouvées « Nb_Zones »

Remarque : On pourra utiliser np.amax(A) pour obtenir le maximum d'un array A.

Dans la suite, les arguments n et T lors de la définition des fonctions seront le nombre de zones « Nb_Zones » et la table « Table ».

Vous devriez obtenir ceci :

```
Nombre de zones:  7
```

On souhaite mettre en place une fonction qui, en un seul parcours de la table T, renvoie une liste LD des données D de chaque zone LD[k]=D=[Taille,Ml,Mc,Col,l_min,c_min,l_max,c_max] avec :

- Taille : nombre de pixels de la zone k (entier)
- Ml : ligne moyenne de la zone k (flottant)
- Mc : colonne moyenne de la zone k (flottant)
- Col : valeur R du triplet RBG de la zone k (0 : Noir, 255 : Blanc)
- l_min,c_min,l_max,c_max sont les limites de la zone k (incluses)

Remarque : le point [Ml,Mc] est le centre de la zone k sur l'image. Pour rappel, on obtient le centre (L,C) d'un ensemble de n pixels (Li,Ci) ainsi : $\begin{pmatrix} M_l \\ M_c \end{pmatrix} = \begin{pmatrix} \frac{1}{n}\sum_{i=0}^{n-1} L_i \\ \frac{1}{n}\sum_{i=0}^{n-1} C_i \end{pmatrix}$

Question 12: Mettre en place la fonction Donnees(T,n,imnb) réalisant le travail attendu et créant la liste « Donnees_Zones »

Vérifier :

```
>>> Donnees_Zones
[[87354, 165.43445062618935, 184.14981569246777, 255, 0, 0, 368, 369], [19182,
314.50000000000045, 115.0, 0, 230, 0, 368, 230], [8649, 91.00000000000006, 91.9
9999999999974, 0, 45, 46, 137, 138], [2162, 252.5, 114.99999999999994, 0, 230,
92, 275, 138], [6348, 68.49999999999997, 253.50000000000003, 0, 0, 231, 137, 27
6], [10765, 262.33887598699613, 327.61922898281347, 0, 184, 277, 322, 369], [20
70, 21.999999999999986, 346.5000000000001, 0, 0, 324, 44, 369]]
```

Dans la suite, l'argument LD lors de la définition des fonctions sera cette liste « Donnees_Zones ».

Nous allons extraire chacune des zones présentes dans l'image en noir et blanc, et nous allons les extraire de l'image noir et blanc et non de l'image en couleur du fait du contexte.
Si pour chaque zone, nous utilisons des slices (`im[l_min:l_max+1,c_max:c_max+1]`), nous allons extraire la zone 1 ainsi :

La zone 3 y est incluse. Ainsi, si des zones se chevauchent (quand on écrit, c'est très probable), on risque de fausser l'apprentissage des caractères. Nous allons donc proposer une fonction de recadrage qui, pour chaque zone, la recadre en ne faisant apparaître que les pixels de la zone concernée.

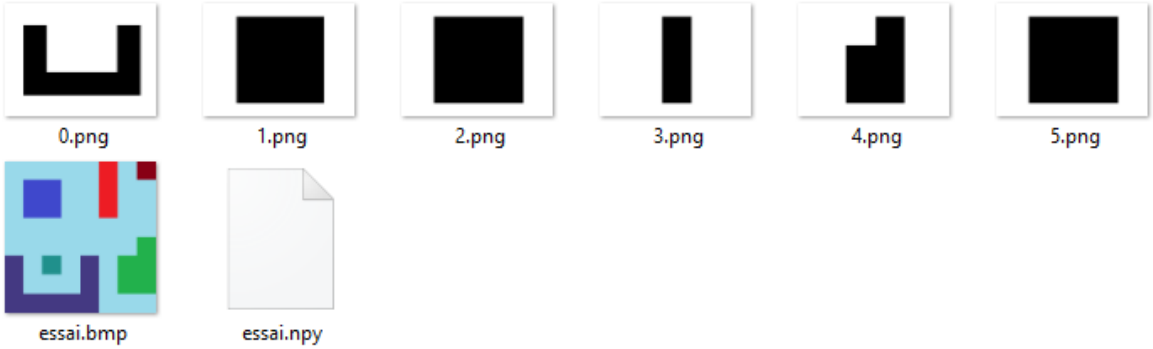
Question 13: Créer la fonction Recadrage(imng,T,LD,k) renvoyant une nouvelle image blanche contenant la portion de im dans les limites (incluses) désignées par les 4 paramètres l_min,l_max,c_min,c_max contenus dans LD

On souhaite maintenant réaliser une fonction qui réalise l'extraction dans l'image étudiée de toutes les zones dont la couleur de l'image noir et blanc est noire (les caractères présents) et dont la taille dépasse un critère Tmin (pour supprimer d'éventuels points isolés), les recadre comme proposé ci-dessus, et les enregistres (plt.imsave sans les afficher) dans un dossier cible au format png. On veillera à afficher dans la console le nombre de zones, les numéros des zones extraites et le nombre d'extractions réalisées.

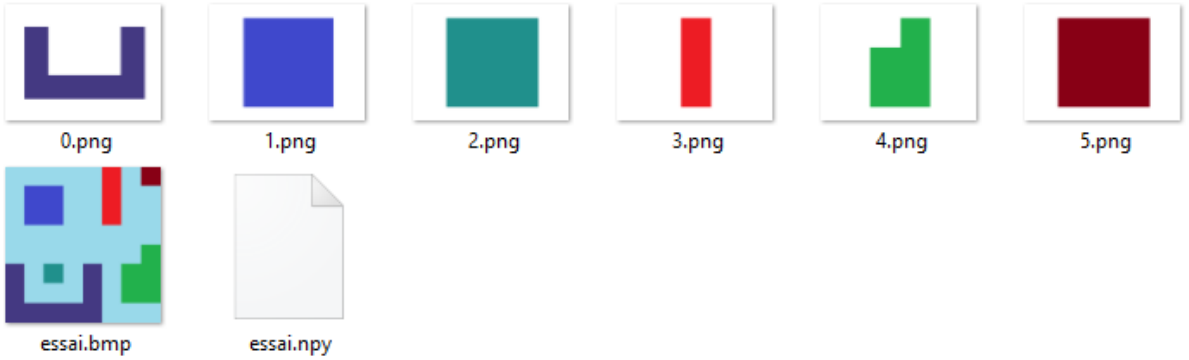
Question 14: Créer la fonction Extraction_Noir(im,T,n,Ld,ch,Tmin) prenant en argument l'image étudiée im, les données issues de la fonction précédente, le chemin du dossier d'enregistrement des images extraites ch et le nombre de pixels minimum Tmin, et réalisant le travail attendu

Utiliser la fonction Extraction_Noir avec Tmin=10 pour enregistrer les zones de l'image étudiée en noir et blanc dans le dossier Essai et vérifier que vous obtenez :

```
Nombre de zones:  7
2/7
3/7
4/7
5/7
6/7
7/7
Nombre d'extractions: 6
```

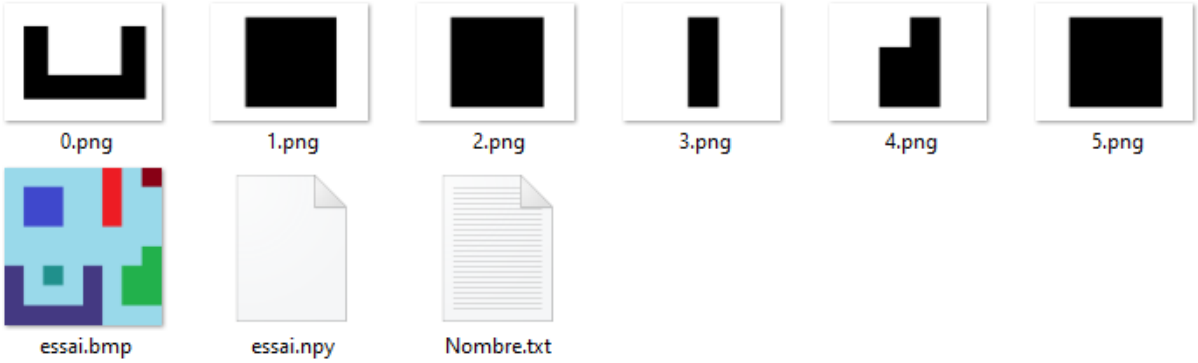


Remarque : il est possible d'extraire très simplement les zones en couleur en mettant l'image Image en argument de l'extraction, mais ce n'est pas attendu dans notre application :



Question 15: Proposer une fonction Ecriture(N,ch) créant (ou remplaçant) un fichier texte dans le dossier indiqué par ch (ex : « ici\\ », avec le nom « Nombre.txt » et enregistrant sur la seul ligne de ce fichier le nombre N

Enregistrer le nombre de zones trouvées dans le dossier « Essai » et vérifier l'apparition du fichier texte.



Exercice 2: Extraction des images

On souhaite maintenant utiliser le travail de l'exercice 1 pour créer un ensemble d'images correspondant à des caractères « appris » à partir d'images contenues dans le dossier « Sources » et aux caractères à identifier dans l'image recherchée dans le dossier « Recherche ». Chacun de ces deux travaux sera réalisé dans un fichier Python différent (2-1 et 2-2) afin de ne pas réapprendre toute la database quand on change d'image recherchée.

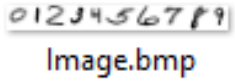
Une database « NIST » existe en ligne et contient une très grande quantité d'images de tous les caractères écrits à la main et est utilisée aujourd'hui dans diverses applications. Elle est téléchargeable ici (environ 1Go compressée avec plus de 1.5 million d'images, près de 75000 « 0 » par exemple) mais ne la chargez pas, je vous fournis un set de départ que vous pourrez alors améliorer. Dans le document « by class », les caractères sont regroupés selon leur type (code ASCII). C'est ce qui nous intéresse. Le code de 0 est 30 et pour 9, c'est 39. Toutes ces images sont déjà en noir et blanc.

Les images de cette database présentent beaucoup de pixels blancs qui vont, si on les garde, allonger leurs temps de traitement. Par ailleurs, l'objectif étant de pouvoir nous même extraire les caractères d'un texte à reconnaitre dans une image, nous mettrons à profit la procédure d'extraction de l'exercice 1 pour :

- Extraire les caractères des images de la database en les redimensionnant au plus juste
- Extraire les caractères de l'image recherchée en les redimensionnant de la même façon

Les données seront ainsi comparables.

J'ai ci-dessous créé à partir d'une image présentant la database, 10 images numérotées de 0 à 9 et nommées « 0.bmp », « 1.bmp » etc., contenant chacune 16 fois chacun des caractères de 0 à 9 (tous différents). Elles sont disponibles dans le dossier « Sources ». J'ai par ailleurs créé une seule image nommée « Image.bmp » mélangeant différents caractères dans le dossier « Recherche ».

Dossier « Sources »	Dossier « Recherche »
 	 Image.bmp 

Remarques

- Grace au critère Tmin, les deux petits défauts à gauche du premier « 2 » et en bas à gauche du 11° « 6 » ne sont pas extraits
- Il a fallu ajouter manuellement deux pixels dans le dernier 5 qui présente un trou

Nous allons proposer une fonction Creation qui, à partir d'une image pleine de caractères au format bmp, en extrait chacun d'entre eux et les enregistre dans un dossier spécifique (format png). Elle sera un agrégat de toutes les fonctions mise en place et utilisées dans l'exercice 1, avec les mêmes paramètres. Les extractions doivent être faites sur les images originales et non les versions en noir et blanc

Question 1: Proposer une fonction Creation(chi,chr) prenant en argument les chemins chi de l'image dans laquelle extraire les caractères et chr du chemin où enregistrer les résultats

On introduit la liste suivante :

```
Caracteres = ['0','1','2','3','4','5','6','7','8','9']
```

Elle servira à la fois à identifier le dossier du même nom contenant les images associées à ce caractère, et à associer à chaque image de la database le caractère contenu dans celles-ci (cf. exercice 3).
On notera donc qu'il serait possible d'ajouter ici des lettres à la suite, de créer les dossiers du même nom dans le dossier « Database » et d'ajouter des images dans « Sources » du même nom contenant plusieurs fois le dit caractère afin d'étendre notre travail à l'alphabet.

Question 2: Utiliser la fonction Creation afin de remplir chacun des 10 dossiers numérotés de 0 à 9 dans le dossier « Database » avec les 16 images du caractère concerné et avec un fichier texte « Nombre.txt » contenant le nombre d'images extraites

Exemple pour le caractère 0 :

Remarquez qu'il serait extrêmement simple d'augmenter la database en ajoutant des images issues de la database NIST aux images bmp sources.

Question 3: En faire de même pour extraire les caractères présents dans l'image du dossier recherche, enregistrant les caractères dans le même dossier ainsi que le fichier texte du nombre de caractères obtenus

Contenu du dossier « Recherche » après exécution du code attendu :

Exercice 3: Outils de création des données

Nous souhaitons dans cette partie avoir à disposition les fonctions qui permettront de créer deux listes de listes, chacune associée soit aux images extraites précédemment de la database, soit aux images de l'image recherchée.

On propose l'import suivant :

```
from skimage.transform import resize
```

Remarque : En cas d'erreur No module named 'skimage', consulter l'annexe en fin de sujet.

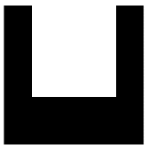
On pourra alors redimensionner une image (format array ouvert avec plt .imread) au format 20x20=2500 pixels sans changement des valeurs des pixels en écrivant :

```
Image = resize(Image,(50,50),anti_aliasing=False,order=0)
```

Attention : L'ouverture avec imread des images au format png renvoie des array contenant des quadruplets de flottants entre 0 et 1, on réalisera les opérations suivantes :

- Redimensionnement en triplets avec « `Image = Image[:,:,:3]` »
- Redimensionnement en 50x50 = 2500 pixels

Question 1: Créer la fonction Lecture_im_resize(Chemin) ouvrant l'image de chemin « Chemin », réalisant les transformations attendues et renvoyant l'array associé



Exemple : `Affiche(100,Lecture_im_resize("Essai\\0.png"))`
Remarque : oui, certains pixels seront gris, les options utilisés dans « resize » ont diminué cet effet, on fera avec. Sinon, il faudrait refaire un traitement nb…
Etant donné que R=G=B=0 (noir) ou R=G=B=1 (blanc) pour chaque pixel, on souhaite créer pour chaque image une liste des valeurs R de tous ses pixels. Pour que ce sujet s'adapte à tout type d'images en couleurs, après avoir extrait les images en couleur auparavant, on stockerait les 3 valeurs R, G et B 😉

Question 2: Proposer une fonction Analyse(Image) prenant en argument l'image « Image » au format array issu de la fonction Lecture_im_resize, et renvoyant sa liste des R transformés en flottants par un parcours ligne par ligne (très important)

Dans notre cas, la liste renvoyée par Analyse aura une taille de 2500. Exemple :

```
>>> len(Analyse(Lecture_im_resize("Essai\\0.png")))
2500
```

On introduit une liste contenant les chemins de plusieurs images. Pour le moment, pour faire un essai, on donne :

```
Liste_Dossier_E = ["Essai\\"]
```

Pour rappel, vous avez dans ce dossier les images extraites de l'exercice 1 en couleur.

Question 3: Proposer une fonction Analyse_Globale(L_Chemin) prenant en argument une liste de chemins de différentes images, réalisant l'analyse de chacune d'entre-elles et renvoyant une liste de toutes les listes d'analyse obtenues

Pour créer les listes des chemins de différentes images contenues dans un dossier, il nous faut lire l'entier contenu dans les fichiers textes générés dans l'exercice 2.

Question 4: Créer la fonction Lecture_txt(ch) ouvrant le fichier texte de chemin « ch » et renvoyant l'entier contenu à sa première ligne

Exemple :

```
>>> Lecture_txt("Essai\\Nombre.txt")
5
```

Question 5: Créer la fonction Liste_Chemin(Ld) qui, à partir d'une liste Ld de dossiers contenant des images, renvoie la liste des chemins de toutes les images de tous les dossiers contenus dans Ld

Exemple :

```
>>> Liste_Chemin(Liste_Dossier_E)
['Essai\\0.png', 'Essai\\1.png', 'Essai\\2.png', 'Essai\\3.png', 'Essai\\4.png']
```

Il nous reste à créer une fonction renvoyant la liste des caractères dans « Caracteres » associés à chacune des images de la liste des chemins renvoyée par « Liste_Chemin » afin d'identifier le caractère contenu dans chacune des images de la database d'apprentissage. Autrement dit, pour chaque dossier Ld[i] d'une liste de dossiers Ld, sera associé le caractère Caracteres[i]. Cette fonction récupèrera le nombre d'image de chaque dossier dans le fichier texte associé.

Question 6: Créer la fonction Liste_Resultats(Ld,Lc) renvoyant la liste des caractères attendue

Question 7: Utiliser les fonctions mises en place pour créer les deux listes Donnees_R (analyse globale des images dans le dossier « Essai ») et Liste_Res (résultats associés à Liste_Resultats sur le dossier « Essai » en utilisant la liste « Caracteres »)

Extrait du résultat dans notre exemple (Liste_Res ne veut rien dire ici) :

```
>>> Liste_Res
['0', '0', '0', '0', '0']

>>> Donnees_R
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
```

Exercice 4: Création des données

Nous allons maintenant utiliser les outils mis en place dans l'exercice 3, et les images extraites dans l'exercice 2.

Données attendues			
Données database		Données recherche	
Liste_Dossier_D	Liste des dossiers des caractères de la database	Liste_Dossier_R	Liste contenant le dossier « Recherche »
Liste_Chemin_D	Liste des chemins de toutes les images de la database	Liste_Chemin_R	Liste des chemins de toutes les images à rechercher
Donnees_D	Liste des listes des données associées à chaque image de la database	Donnees_R	Liste des listes des données associées à chaque image recherchée
Resultats_D	Liste des caractères de chaque image de Donnees_D		

Question 1: Dans le fichier 4-1, écrire les lignes de code créant les données associées aux images de la base de données

Extraits de ce que j'obtiens :

```
>>> Liste_Dossier_D
['Database\\0\\', 'Database\\1\\', 'Database\\2\\', 'Database\\3\\', 'Database\\
4\\', 'Database\\5\\', 'Database\\6\\', 'Database\\7\\', 'Database\\8\\', 'Datab
ase\\9\\']

>>> Liste_Chemin_D
['Database\\0\\0.png', 'Database\\0\\1.png', 'Database\\0\\2.png', 'Database\\0\
\3.png', 'Database\\0\\4.png', 'Database\\0\\5.png', 'Database\\0\\6.png', 'Data
base\\0\\7.png', 'Database\\0\\8.png', 'Database\\0\\9.png', 'Database\\0\\10.pn
>>> Donnees_D
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
>>> Resultats_D
['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2',
'3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3',
'4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4', '4',
'5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5', '5',
'6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6', '6',
'7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7', '7',
'8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8', '8',
'9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9', '9']
```

Question 2: Dans le fichier 4-2, écrire les lignes de code créant les données associées aux images recherchées

Extraits de ce que j’obtiens :

```
>>> Liste_Dossier_R
['Recherche\\']

>>> Liste_Chemin_R
['Recherche\\0.png', 'Recherche\\1.png', 'Recherche\\2.png', 'Recherche\\3.png',
'Recherche\\4.png', 'Recherche\\5.png', 'Recherche\\6.png', 'Recherche\\7.png',
'Recherche\\8.png', 'Recherche\\9.png']

>>> Donnees_R
[[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0
```

Exercice 5: Apprentissage

Nous voici arrivée à l'étape d'apprentissage supervisé qui nous permettra ensuite de reconnaitre n'importe quelle image.
Dans un premier temps, et pour commencer à partir de cette partie 5, procurez-vous le « Dossier élèves 5 à 6 ». Sinon, continuez avec vos codes des parties précédentes.
En cas d'erreurs avec le module skimage, consulter l'annexe en fin de sujet.
Nous allons étudier deux méthodes d'intelligence artificielle que sont la méthode knn et la méthode des réseaux de neurones.

Méthode knn	Cette méthode ayant été programmée en informatique du tronc commun (ITC2), rappelons simplement son principe. La phase d'apprentissage permet d'apprendre toutes les données de la database associées aux images des différentes familles d'images classées par caractères. Ensuite, pour chaque nouvelle image im, une distance euclidienne est calculée entre im et toutes les images de la base de données. On sélectionne alors les k plus proches images de im en les triant par ordre croissant des distances obtenues. On détermine enfin la famille d'appartenance de im en identifiant la famille majoritaire dans les k images les plus proches.
Méthode rn « Réseau de Neurones »	Nous allons utiliser la méthode MLP (Multi-layer perceptron), c'est-à-dire un réseau de neurones multicouches. Un neurone (ou perceptron) est un outil informatique basé sur l'analyse du fonctionnement biologique des neurones réels, qui, à n entrées x_i $\{x_1, x_2, \dots, x_n\}$, calcule une somme f (fonction d'agrégation) utilisant des poids ω_i sur chacune de ces entrées en utilisant un biais b $(f = \sum x_i \omega_i + b)$ et retourne une valeur en sortie à l'aide d'une fonction d'activation, par exemple $\begin{matrix} s = 1 \; si \; f \geq 0 \\ s = 0 \; si \; f < 0 \end{matrix}$. Avec cette fonction d'activation, l'utilisation d'un seul neurone permet de départager les données de manière linéaire. On utilise alors des fonctions d'activation non linéaires et des neurones en parallèle sur plusieurs couches (couche d'entrée, couches cachées et couche de sortie, les résultats de chaque couche étant transmis à une couche inférieure) afin d'étendre les possibilités de prédiction de résultats sur des données très complexes. On utilise ensuite des algorithmes de renforcement (ex : descente de gradients) permettant d'entrainer le réseau de neurones sur des données de référence en affinant tous les poids ω_i et le biais b. Voici une chaine youtube très bien faite pour aller plus loin si vous le désirez. La première vidéo suffit à comprendre les bases que nous utilisons ici.

Nous allons utiliser le module « sklearn » sous Python (site) pour réaliser ces méthodes. Quelle que soit la méthode utilisée, les étapes sont les mêmes :

- Séparation des données en deux paquets (paquet d'entrainement « train », et paquet de test « test »)
- Création de l'outil
- Apprentissage des données « train »
- Vérification du score sur les données « test »

Si besoin, pour installer le module, tenter dans la console « `pip install scikit-learn` » ou « `python -m pip install scikit-learn` ».

On effectue l'import suivant (documentation) :

```
from sklearn.model_selection import train_test_split
```

On écrit alors :

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0)
```

Cette instruction sépare les données x (remplacer `x` par les données des images de la database) et y (remplacer `y` par les caractères espérés pour chaque image issue des données) en deux familles, « train » et « test ». Sans aucune précision, 75% des données servent pour « train » et 25% pour « test ». L'option « `random_state` » à définir avec un entier positif permet d'obtenir un découpage aléatoire constant des données quelle que soit l'exécution du code (afin d'obtenir les mêmes prévisions). S'il n'est pas précisé, chaque exécution de l'apprentissage utilisera une répartition des données différente.

On importe ensuite un outil qui permettra de créer, soit un réseau de neurones, soit une résolution knn :

```
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
```

On créer alors le modèle de réseau de neurones ou le modèle knn :

```
rn = MLPClassifier(solver='à choisir',max_iter=10000,random_state=0)
knn = KNeighborsClassifier(n_neighbors=5)
```

Quelques explications :

Modèle rn	Modèle knn
Documentation	Documentation
On peut utiliser différents solvers (adam, sgd, lbfgs)	Il est très simple de comprendre l'option : `n_neighbors=5` On prend dans cet exemple les 5 plus proches voisins
Par défaut, `max_iter` est défini à 200. S'il n'est pas précisé dans notre exemple, un message indiquera que l'optimisation n'a pas convergé en 200 itérations	
`random_state` doit être défini avec un entier positif et garanti un apprentissage identique à chaque exécution	

Il reste alors, quel que soit le modèle utilisé (knn, rn), à écrire :

```
methode.fit(x_train,y_train)
Score = methode.score(x_test,y_test)
Score = int(Score*10000)/100
print("Score:",Score,"%")
```

Quelques explications :

- Remplacer le mot « methode » par « knn » ou « rn »
- fit : phase d'apprentissage de la méthode utilisée
- score : renvoie le score obtenu sur les données test

Question 1: Ecrire les lignes de code afin de séparer les données en deux sets « train » et « test »

Question 2: Ecrire les lignes de code permettant de créer les modèles associés aux méthode knn et rn, de réaliser les apprentissages et d'afficher les scores dans la console

Remarque : on choisira le solveur de rn donnant le meilleur résultat

Vous devriez obtenir :

```
Score: 77.5 %
Score: 65.0 %
```

Remarque : il arrive que vous obteniez d'autres résultats, faites avec, je n'ai pas encore l'explications (ex : 65%, 72,5%, 75%)…

Une fois l'apprentissage terminé, on peut l'enregistrer ou le charger ainsi, quelle que soit le modèle utilisé (rn, knn) :

```
from joblib import dump, load
dump(methode, 'Modèle_XXX.joblib')
methode = load('Modèle_XXX.joblib')
```

Ayez conscience que l'enregistrement puis le chargement des modèles n'est pas obligatoire.

Exercice 6: Prédiction

Notre réseau de neurones « rn » est créé, et notre modèle « knn » de même. Il ne reste plus qu'à leur demander d'identifier chacune des images recherchées à l'aide de la liste de leurs données.

Quelle que soit le modèle (rn, knn), on procède ainsi :

```
Sol = methode.predict([X])[0]
```

X étant la donnée recherchée, Sol sera le caractère reconnu.

Question 1: Proposer deux fonctions Prediction_rn(X) et Prediction_knn(X) renvoyant la prédiction associée à X avec le modèle concerné

On souhaite dans un premier temps étudier les matrices de confusion de nos deux modèles sur les données « test ». On propose le code suivant :

```
from sklearn.metrics import confusion_matrix
def Confusion(f,x,y_true):
    y_pred = []
    for X in x:
        y_pred.append(f(X))
    Mat = confusion_matrix(y_true,y_pred)
    return Mat
```

Avec f la fonction de prédiction, x la liste des données à prédire, et la liste y_true les valeurs attendues de la prédiction.

Question 2: Mettre en place le code créant les matrices de confusion Mat_rn et Mat_knn des deux modèles et les affichant dans la console

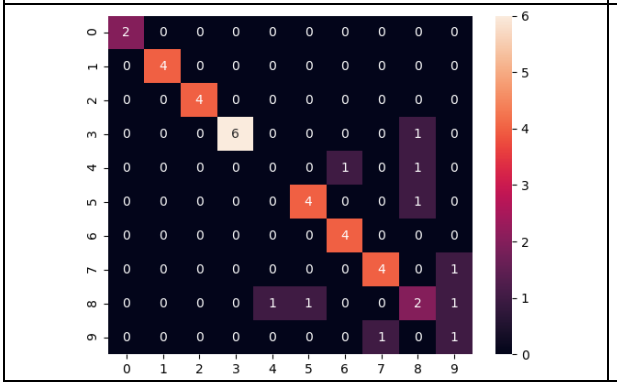
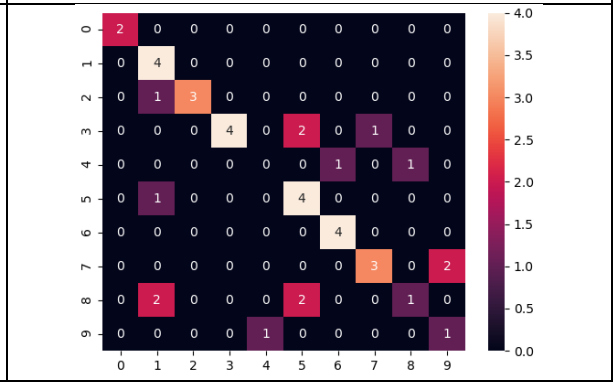
RN	KNN
[[2 0 0 0 0 0 0 0 0 0 0] [0 4 0 0 0 0 0 0 0 0 0] [0 0 4 0 0 0 0 0 0 0 0] [0 0 0 6 0 0 0 0 1 0] [0 0 0 0 0 0 1 0 1 0] [0 0 0 0 0 4 0 0 1 0] [0 0 0 0 0 0 4 0 0 0] [0 0 0 0 0 0 0 4 0 1] [0 0 0 0 1 1 0 0 2 1] [0 0 0 0 0 0 0 1 0 1]]	[[2 0 0 0 0 0 0 0 0 0 0] [0 4 0 0 0 0 0 0 0 0 0] [0 1 3 0 0 0 0 0 0 0 0] [0 0 0 4 0 2 0 1 0 0] [0 0 0 0 0 0 1 0 1 0] [0 1 0 0 0 4 0 0 0 0] [0 0 0 0 0 0 0 4 0 0 0] [0 0 0 0 0 0 0 0 3 0 2] [0 2 0 0 0 2 0 0 1 0] [0 0 0 0 1 0 0 0 0 1]]

Le code suivant affiche la matrice de confusion :

```
import seaborn
import pandas
def Affiche_Confusion(Mat):
    Nl,Nc = Mat.shape
    plt.figure()
    DetaFrame_cm = pandas.DataFrame(Mat,range(Nl),range(Nc))
    seaborn.heatmap(DetaFrame_cm,annot=True)
    plt.show()
```

En cas d'erreur « No module named 'seaborn' », installer le module : conda install seaborn

Question 3: Afficher les matrices de confusion des deux modèles à l'aide de la fonction « Affiche_Confusion ».

RN	KNN
	

Question 4: Mettre en place les lignes de code recréant le message contenu dans l'image recherchée et l'affichant dans la console pour la méthode rn

Résultat :

```
Texte: 0173456709
```

Question 5: En faire de même pour la méthode knn

Résultat :

```
Texte: 0170456714
```

Il est tout à fait normal que l'algorithme mis en place ne soit pas très efficace, nous n'avons finalement appris que 12 images (75%) de chaque caractère.

Exercice 7: Pour aller plus loin

Il ne reste plus qu'à agrandir considérablement la database à l'aide de « NIST » de deux manières relativement simples : Créer une procédure sous Python qui :

1- Ouvre les images bmp dans « Source », les agrandit et y ajoute les uns après les autres des caractères de NIST (ou pour seulement quelques images, le faire à la main)
2- Ouvre chaque image de NIST, utilise une extraction similaire à celle de l'exercice 1 par image et en extrait les caractères dans les dossiers des caractères associés dans « database »

Dans tous les cas, on enregistre les images dans les dossiers déjà remplis lors de l'exercice 2, en complétant bien les dossiers des fichiers textes associés.

Attention : L'extraction que nous avons réalisée nécessite que chaque caractère soit défini en une seule zone contiguë de pixels. Deux choix s'offrent à nous :

A- Comme chaque image ne contient qu'un seul caractère, on pourrait redimensionner les images aux valeurs min et max des pixels noirs, on garderait ainsi des caractères en plusieurs morceaux, mais… Certaines images ont des morceaux de caractères voisins, je ne les veux pas

B- Filtrer les caractères en n'extrayant que ceux qui ne présentent qu'une seule grande zone noir. Autrement dit, pour chaque image contenant théoriquement un seul caractères, trier les zones par taille et extraire la première zone noire s'il n'y a qu'une seule grande zone noire

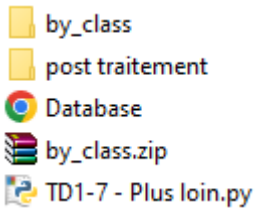
Enfin, on pourrait envisager autre chose, en apprenant directement les caractères de la database sans les transformer, mais il faudra alors revoir la procédure d'extraction dans l'image recherchée afin de mettre en forme les images extraites comme celles de la database (dimensions image et dimension caractère), ce que je n'envisage pas.

J'ai proposé la méthode 2B dans la suite, en supprimant aussi quelques images corrompues.

Vous avez à disposition un fichier « Extractions.py » que je ne l'expliquerai pas ici (vous avez tout maintenant pour le comprendre), qui permet de réaliser autant d'extractions que souhaité dans la database NIST, afin d'augmenter le nombre d'images apprises.

Quelques précisions :

- Le mettre à la racine de la database, en parallèle du dossier « by_class » extrait de « by_class.zip » téléchargé ici
- Créer un dossier « post traitement »
- Définir le nombre d'images à extraire de chaque caractère dans la liste Lni, sans dépasser les valeurs dans Ln
- Exécuter l'exercice 1 puis faire tourner le 7 😊



Ensuite, collez votre code « 1 » dans le même dossier, changer « Database » pour « post traitement » et faites-le tourner pour créer les données associées aux images obtenues. Le reste devrait vous paraître simple…

Je vous partage la base de données des images post traitées et les modèles RN et KNN que j'ai pu établir avec cette base de données en extrayant 9 628 (les 1000 premières de chaque dossier), puis les 334 124 images retenues dans toute la base des chiffres, 75% servant à l'apprentissage et 25% aux tests, qui vous permettront d'améliorer la reconnaissance pratiquée précédemment :
Pour résumer ce qu'ils font : ils sont capables de reconnaitre une image en noir et blanc d'un caractère entre 0 et 9 recadré au plus près et de dimensions 50x50 pixels.

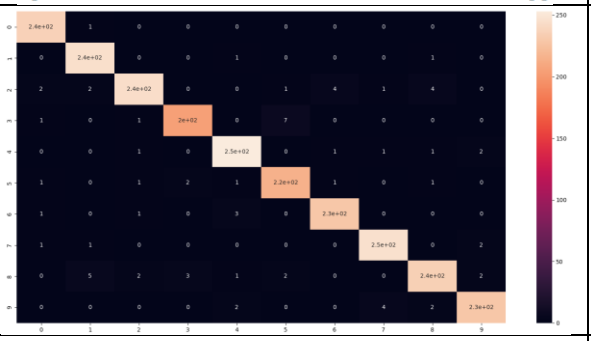
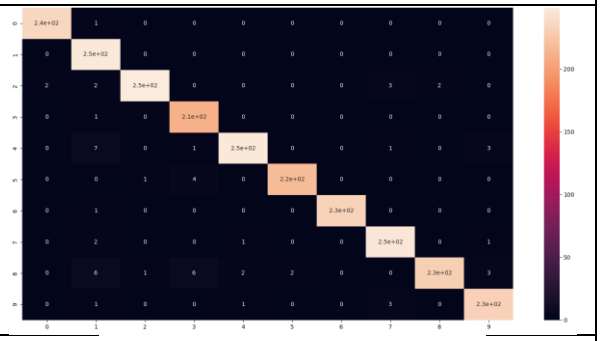
Remarque : Pour les 334 124 images, compter 2 jours d'extraction, 3h d'analyse globale, 3h d'apprentissage et 26h de tracé des matrices de confusion.

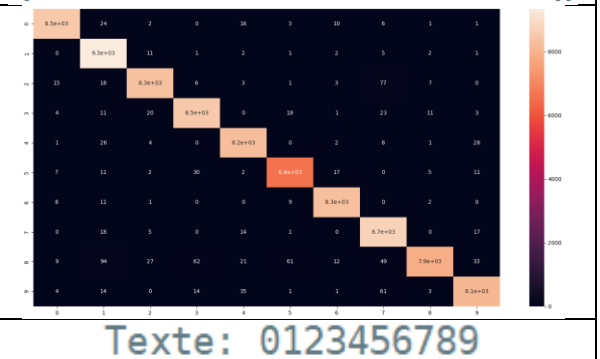
Question 1: Charger l'un de ces modèles et les essayer sur les images recherchées

Petite précision : pour créer l'image recherchée « Image », j'ai pris des images de caractères dans des dossiers autres que « train ». Il n'y a donc pas de triche.

Si besoin, vous avez à disposition le minimum nécessaire « Starter kit » pour la reconnaissance d'images quelconque (modèle rn sur les 334 124 images, code préparé et images recherchées).

Résultats

<table>
<tr><td colspan="2">Résultats obtenues sur la base de 9 628 images</td></tr>
<tr><td>RN</td><td>KNN</td></tr>
<tr><td>Score: 97.05 %</td><td>Score: 97.59 %</td></tr>
<tr><td>[[236 1 0 0 0 0 0 0 0 0]
[0 245 0 0 1 0 0 0 1 0]
[2 2 244 0 0 1 4 1 4 0]
[1 0 1 204 0 7 0 0 0 0]
[0 0 1 0 253 0 1 1 1 2]
[1 0 1 2 1 215 1 0 1 0]
[1 0 1 0 3 0 229 0 0 0]
[1 1 0 0 0 0 0 246 0 2]
[0 5 2 3 1 2 0 0 235 2]
[0 0 0 0 2 0 0 4 2 229]]</td><td>[[236 1 0 0 0 0 0 0 0 0]
[0 247 0 0 0 0 0 0 0 0]
[2 2 249 0 0 0 0 3 2 0]
[0 1 0 212 0 0 0 0 0 0]
[0 7 0 1 247 0 0 1 0 3]
[0 0 1 4 0 217 0 0 0 0]
[0 1 0 0 0 0 233 0 0 0]
[0 2 0 0 1 0 0 246 0 1]
[0 6 1 6 2 2 0 0 230 3]
[0 1 0 0 1 0 0 3 0 232]]</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Texte: 0123456789</td><td>Texte: 0173456789</td></tr>
<tr><td>Taille du modèle : 4 Mo</td><td>Taille du modèle : 141 Mo</td></tr>
</table>

<table>
<tr><td colspan="2">Résultats obtenues sur la base de 334 124 images</td></tr>
<tr><td>RN</td><td>KNN</td></tr>
<tr><td>Score: 98.98 %</td><td>Score: 98.74 %</td></tr>
<tr><td>[[8495 9 7 6 8 5 13 2 11 3]
[3 9322 16 5 2 7 5 5 5 1]
[12 16 8332 14 7 8 16 18 18 2]
[3 3 30 8472 0 32 1 18 28 8]
[3 1 2 0 8272 2 10 10 3 12]
[8 2 4 25 4 6606 14 5 9 7]
[10 2 5 0 8 15 8275 0 3 0]
[1 9 15 4 15 0 0 8739 1 17]
[8 38 17 21 12 32 8 11 8058 18]
[5 2 3 5 39 10 2 32 15 8109]]</td><td>[[8496 24 2 0 16 3 10 6 1 1]
[0 9346 11 1 2 1 2 5 2 1]
[15 18 8313 6 3 1 3 77 7 0]
[4 11 20 8504 0 18 1 23 11 3]
[1 26 4 0 8245 0 2 8 1 28]
[7 11 2 30 2 6599 17 0 5 11]
[8 11 1 0 0 9 8287 0 2 0]
[0 18 5 0 14 1 0 8746 0 17]
[9 94 27 62 21 61 12 49 7855 33]
[4 14 0 14 35 1 1 61 3 8089]]</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Texte: 0123456789</td><td>Texte: 0123456789</td></tr>
<tr><td>Taille du modèle : 4 Mo</td><td>Taille du modèle : 5 Go</td></tr>
</table>

On voit ici tout l'intérêt du réseau de neurones qui donne une « tête bien faite » plutôt que knn qui donne une « tête bien pleine ».

Annexe

Erreur sur skimage ? Tenter cette solution :

```
install conda
conda install scikit-image
```

Ou encore :

```
pip install --upgrade pip
pip install scikit-image
```