TP n°4: Traitement de données réelles

Le compte-rendu doit être rédigé en Jupyter-Notebook, Markdown ou LaTeX. Il doit comporter à la fois les codes mais également **et surtout** *des commentaires et interprétations clairs et pertinents concernant (i) vos choix, (ii) vos résultats en rapport avec les notions vues en cours.*

1 Descriptif du TP

Avec ce quatrième TP, nous entrons dans le cœur applicatif du projet. Après avoir appris à construire des algorithmes d'apprentissage, il s'agit maintenant de les mettre en œuvre sur des données réelles. Cela va nous demander un important travail de pré-traitement des données réelles pour qu'elles ressemblent au mieux aux données apprises par l'algorithme.

Spécifiquement, nous nous intéresserons à la reconnaissance de chiffres *manuscrits par vos soins*, ou même extraits d'un document personnel rempli à la main (numéro de sécurité sociale, date de naissance, etc.). Pour cela, nous procèderons comme suit:

- nous allons tracer un ensemble de chiffres manuscrits entre 0 et 1, puis entre 0 et 9, au stylo foncé (noir ou bleu foncé) sur une feuille quadrillée (ou une feuille blanche sur laquelle vous délimiterez des cases horizontales et verticales régulières au crayon à papier)
- cette feuille sera ensuite proprement scannée (ou très méticuleusement prise en photo) afin de générer une image `PNG` ou `JPG` que nous ouvrirons sur Python
- en se rappelant que les données `MNIST` sont tracées en blanc (valeurs proches de 255) sur fond purement noir (valeurs égales à 0) tandis que le scan sera de contraste opposé sur un fond imparfaitement blanc et en couleur, nous allons devoir pré-traité l'image pour que les chiffres soient identifiables par notre algorithme d'apprentissage
- une fois pré-traitée, l'image des chiffres sera "découpée une liste de boîtes" manuellement définies par vos soins et qu'il s'agira alors de reconnaitre, après mise à l'échelle en 28×28 pixels, grâce aux algorithmes développés au cours des TP1, TP2 et TP3.

Le champ des solutions et des bonnes idées pour le pré-traitement est assez large, de sorte que la compétition d'intelligence artificielle débute véritablement aujourd'hui!

2 Mise en place

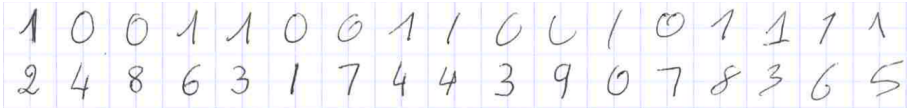
2.1 Un peu de bricolage

Comme mentionné ci-dessus, préparez tout d'abord, sur feuille quadrillée ou non (mais dans ce cas, tracez des lignes régulières au crayon fin) deux lignes de chiffres régulièrement espacés et de même taille:

- la première ligne contiendra uniquement des 0 et des 1
- la deuxième ligne contiendra des chiffres de 0 à 9

Quelques conseils de réalisation:

- attention au chiffre 7 qu'on devra écrire à l'américaine, sans la barre centrale
- pour rendre l'exercice plus intéressant, on pourra s'attacher à écrire les chiffres très proprement en début de ligne et plus difficilement lisibles en fin de ligne (de sorte à pouvoir tester la puissance de l'algorithme).



2.2 Pré-traitement des données

Scannez ou prenez précautionneusement une photo de votre feuille manuscrite. Enregistrez le fichier, disons sous le nom `digits.png`, et chargez le à l'aide de la commande `imread()` de la librairie `matplotlib.image` comme suit:

```
import matplotlib.image as img
digits = img.imread('digits.png')
```

Vérifiez que votre image est correctement chargée en utilisant la fonction `imshow()` de la librairie `matplotlib.pyplot`.

La variable `digits` contient ici une ou plusieurs matrices contenant la valeur des pixels de l'image: une seule matrice si l'image est en niveaux de gris, mais plus souvent plusieurs matrices des couches de couleurs (rouge, verte et bleue, par exemple). Utilisez la fonction `shape()` de `numpy` pour obtenir cette information. Dans le cas de plus d'une couche, nous allons fusionner les couches en une seule en les moyennant grâce à la fonction `mean` de `numpy`:

```
digits = np.mean(digits,2)
```

où le paramètre 2 réalise ici la moyenne sur la troisième dimension (numérotée 2) de `digits`. Vérifiez à nouveau les dimensions de `digits` après moyennage et confirmez qu'il ne reste plus qu'une seule matrice. Vérifiez par la même occasion que les entrées de `digits` sont dans l'intervalle $[0, 1]$, et sinon pensez à les normaliser!

Visualisez à nouveau l'image grâce à `imshow()`: on pourra utiliser l'option `cmap='gray'` (on rappelle que `cmap` est la carte des couleurs) pour une meilleure visualisation.

Le problème majeur qui se pose maintenant est que l'image a un fond blanc (donc associée à des valeurs proches de 1) tandis que les chiffres ont une couleur sombre (proche de 0). Proposez une solution simple qui permettrait "d'inverser" les couleurs tout en conservant les valeurs dans $[0, 1]$.

Astuce: on pourra utiliser la propriété de "propagation" de Python qui permet d'effectuer une opération algébrique simultanément sur tous les éléments d'un vecteur ou d'une matrice (par exemple, `4+X` *ajoute* `4` *à toutes les entrées de* `X`*, ou encore* `X>0.1` *renvoie une matrice composée de uns aux entrées de* `X` *supérieures à* 0.1 *et de zéros sinon).*

Cette solution nous rapproche des images de la base d'entrainement de `MNIST` mais pas entièrement car le fond de l'image n'est pas uniformément noir (égal à 0), ce qui va évidemment troubler la classification. En utilisant à nouveau l'astuce de propagation, proposez une solution qui nous permet (i) de transformer l'image `digits` en un ensemble de pixels noirs et blancs ou (ii) de ne conserver que les valeurs de `digits` supérieures à un seuil (les pixels les plus blancs) et met les autres à zéro (le fond noir).

2.3 Découpage et mise à l'échelle

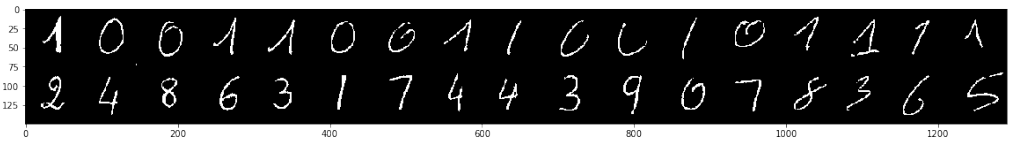
Nous allons maintenant procéder au découpage des chiffres manuscrits et à leur mise en forme (redimensionnement au format 28×28 pixels). Dans un prochain TP, nous automatiserons évidemment cette tâche et laisserons la machine trouver et identifier elle-même les chiffres.

Rappelons tout d'abord que "l'image" `digits` n'est en fait qu'une matrice de valeurs (qui encode le niveau de gris de chaque pixel). Récupérer une partie de l'image revient donc à sélectionner un sous-ensemble des coordonnées de la matrice: par exemple, `digits[4:9,6:20]` extrait de `digits` la sous-matrice rectangulaire de coordonnées lignes 4 à 8 (rappelons que la dernière coordonnée n'est pas prise) et colonnes 6 à 19. Grâce à la visualisation par `imshow()`, vous avez accès à ce système de coordonnées. Identifiez les coordonnées de votre premier chiffre manuscrit et visualisez la sous-matrice correspondante grâce à `imshow()`. Procédez de même avec les chiffres suivants et, du fait de la régularité de répartition des chiffres sur votre feuille, automatisez l'identification des positions des chiffres: on pourra par exemple relever (i) la taille `size`

en pixels des zones d'intérêt, (ii) la position `top_left` du coin en haut à gauche de la première sous-matrice et (iii) itérer sur les zones grâce à une boucle `for`. Affichez alors toutes les images automatiquement découpées, par exemple dans un multi-graphe comme suit:

```
size = ...
top_left = [... , ...]
figure, axis = plt.subplots(1,10,figsize=(20,20))
for index in range(...):
  axis[index].imshow(digits[... : ... , ... : ...],cmap='gray')
```

Vous devriez obtenir un graphe similaire au suivant:



Très vraisemblablement, la longueur `size` des carrés d'intérêt ne vaut pas 28 comme l'attend un classifieur entraîné sur la base d'apprentissage `MNIST`. Nous devons donc maintenant passer les sous-matrices à l'échelle. Matriciellement, c'est une opération très délicate à mener: à moins que `size` soit un multiple exact de 28, il n'existe pas de manière naturelle de transformer un "tableau de chiffres" en un tableau plus petit. Nous allons donc temporairement repasser dans le monde des images dans lequel existent de nombreuses méthodes de réduction de dimension avec des effets de moyennage, lissage, etc. Pour cela, nous allons importer la fonction `resize` de la bibliothèque `skimage.transform` qui redimensionne une matrice d'image `input` en une matrice d'image `output` aux dimensions `(dim_x,dim_y)` comme suit:

```
from skimage.transform import resize
output = resize(input,(dim_x,dim_y))
```

Affichez à nouveau l'ensemble des chiffres après compression de taille pour confirmer qu'ils sont bien au bon format.

Vous êtes maintenant prêts à lancer la classification automatique!

3 Classification

Cette section est volontairement laissée très libre, afin de mener un réel travail d'ingénieur en informatique. Il s'agit ici de reprendre les classifieurs développés au cours des TP1, TP2 et TP3, et de les appliquer à chacune des images pré-découpées dans la section précédente. Vous avez désormais tous les outils pour y parvenir.

L'objectif est bien sûr d'identifier les chiffres manuscrits mais **aussi et surtout** de mener un travail minutieux d'amélioration de la qualité des classifieurs:

- dans un premier temps, un classifieur binaire sur les chiffres 0 et 1 uniquement
- une fois validé, dans un second temps, un classifieur sur les dix chiffres de 0 à 9.

Afin de suivre l'amélioration des performances sur une si petite base expérimentale, il sera pertinent d'afficher (par exemple dans un graphe) les probabilités fournies par la fonction `predict_proba` afin de vérifier que l'identification des chiffres manuscrits est nette.

Important: Tâchez de rendre votre code le plus "souple" et automatisé possible, en définissant un maximum de variables d'option (choix du classifieur, utilisation de la représentation HOG, paramètres divers, position `top_left` de l'image scannée, etc.) afin que votre code puisse être facilement adapté à un nouveau jeu de données, en évitant toute redondance, et en commentant le code au maximum.

Savoir développer un code qui marche, c'est bien, mais **savoir développer un code agréable et réutilisable par les autres**, c'est la marque des futurs ingénieurs talentueux!

Exemple de sortie visuelle du TP: