

## Pipeline complète de classification d'images

5IC-IG4 Vision par ordinateur

---

A rendre individuellement, le lien du repo git contenant à minima :

- Le code python exécutable sous la forme d'un jupyter notebook ou d'un fichier python avec cellules interactives
  - Un README.md avec les réponses aux questions
  - Un requirements.txt avec toutes les dépendances nécessaires à l'exécution du code
- 

### 1- Mise en place de l'environnement de code

- a) Mettre en place un repo git, un virtualenv
- b) Créer un jupyter notebook (.ipynb) ou un script python interactif (.py avec des cellules délimitées par `#%%`). Importer les dépendances nécessaires à leur utilisation.
- c) Installer et importer les bibliothèques utiles

- i. Numpy  
`import numpy as np`
- ii. sklearn
- iii. matplotlib  
`import matplotlib.pyplot as plt`  
`import matplotlib.image as img`
- iv. opencv  
`import cv2`

- d) Manipuler les bibliothèques utiles

Numpy

- i. Créer une liste, X, de 1000 points avec valeur aléatoire dans l'intervalle [0, 3]  
On utilisera `np.random.rand`
- ii. Calculer la moyenne, l'écart type et la médiane de cette liste. Arrondir dans le code les valeurs au centième. Noter les valeurs  
Les fonctions utiles sont `np.mean`, `np.std`, `np.median`, et `round`
- iii. Créer une liste, X\_bis, de 1000 points avec valeur aléatoire dans l'intervalle [0, 3]
- iv. Calculer la moyenne, l'écart type et la médiane de cette nouvelle liste.
- v. Comparer les résultats de moyenne, écart type et médiane des listes X et X\_bis.
- vi. Fixer l'aléa pour pouvoir reproduire les résultats et vérifier que la cellule donne maintenant des valeurs constantes

On pourra le faire avec : `np.random.seed`


💡 Pour la suite, on se rappellera que les modèles d'apprentissage machine ne sont pas déterministes, il est donc important de fixer l'aléa pour avoir des résultats reproductibles.

- vii. Créer une liste, y, de 1000 points ayant la valeur de  $\sin(X)$  auquel on ajoute un bruit gaussien aléatoire ayant une amplitude de 10% (0.1)  
On pourra créer une liste de 1000 points de bruit gaussien aléatoire avec : `noise = np.random.randn(1000)`

Matplotlib

- viii. Visualiser y en fonction de X sous forme de graph 'scatter'  

```
plt.figure(figsize=(8,6))
plt.scatter(X, y)
plt.show()
```

 Pour la suite, on oubliera pas d'utiliser `plt.show()` après chaque plot matplotlib.
- ix. Changer la taille de la figure
- x. Visualiser le bruit gaussien, noise, sous forme d'histogramme. Le nombre de bins est fixé à 50.  
 Le type de plot à utiliser est : `plt.hist`
- xi. A quelle fonction la distribution de noise fait penser ?

## 2- Données

- a) Télécharger les images du dossier data1  
[https://drive.google.com/file/d/1lYns1282S2M898l7l6njrh8SMM1sqlFh/view?usp=drive\\_link](https://drive.google.com/file/d/1lYns1282S2M898l7l6njrh8SMM1sqlFh/view?usp=drive_link)
- b) Informations générales sur les données
  - i. Combien y a-t-il d'images ? (On ne les compte pas à la main !)
  - ii. Quel est le format et la taille des images ?
- c) Visualiser les données
  - i. Visualiser une des images en couleur  

```
image = img.imread(<folder_path>)
```
  - ii. Visualiser la même image en noir et blanc  
 On utilisera `plt.imshow(image[:, :, 1], cmap="gray")`
  - iii. Visualiser la même image à l'envers  
 On changera l'argument `origin` de `plt.imshow`
- d) Homogénéiser les images : à la fin de cette étape, nous avons deux numpy array contenant notre dataset. C'est-à-dire, `images` contenant toutes les images à la même dimension, `labels` contenant les labels (bike ou car) dans le même ordre des images de `images`.
  - i. Définir les chemins aux dossiers bike et car  

```
bike_folder = <your_relative_path>
car_folder = <your_relative_path>
```
  - ii. Définir la taille voulue  

```
target_size = (224,224)
```
  - iii. Créer une méthode qui prend en entrée le dossier à considérer et ressort les images à la bonne dimension avec la liste des labels correspondants.  

```
def peuplate_images_and_labels_lists(image_folder_path):
    images= []
    labels = []
    for filename in os.listdir(image_folder):
        image = cv2.imread(os.path.join(image_folder, filename))
        # TODO: TO COMPLETE
```
  - iv. Créer des array numpy pour les images et les labels  
 On utilisera la méthode de la fonction précédente. Pour passer des listes aux array numpy, on pourra utiliser `np.array(<your_list>)`
- e) Preprocessing des images : représentation des images par la création d'une liste de taille (`nb_image*nb_features`)  

```
images = np.array([image.flatten() for image in images])
```
- f) Séparation des sets d'entraînement et de test

- i. Importer la méthode adaptée  
On utilise : la méthode `train_test_split` de `sklearn.model_selection`
- ii. Séparer les sets  
On utilise 80% des images pour l'entraînement et 20% des images pour le test.  
`X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=0)`
- iii. A quoi sert l'argument `random_state` ?

### 3- Modèles de classification

- a) Premier modèle de classification avec sklearn : arbre de décision
  - i. Importer la classe `DecisionTreeClassifier` de `sklearn.tree`
  - ii. Définir l'arbre de décision  
`clf = DecisionTreeClassifier(random_state=0)`
  - iii. Entraîner l'arbre de décision
  - iv. Comment prédire le label de la première image du set de test ?
- b) Deuxième modèle de classification avec sklearn : Support Vector Machine (SVM)  
Suivre les mêmes étapes que précédemment avec la classe `SVC` de `sklearn.svm`
- c) Evaluation
  - i. Accuracy  
Importer la méthode `accuracy_score` de `sklearn.metrics`  
Calculer l'accuracy du modèle 1  
Calculer l'accuracy du modèle 2
  - ii. Matrice de confusion  
Importer la méthode `confusion_matrix` de `sklearn.metrics`  
Calculer la matrice de confusion du modèle 1  
Interpréter la matrice de confusion : combien de bike ont été classifiés comme des car ? Combien de car ont été classifiés comme des bike ?  
Calculer la matrice de confusion du modèle 2
  - iii. Bonus : Calculer la précision, spécificité (recall) et tracer la courbe ROC avec le modèle 1

### 4- Comparaison de pipeline et fine tuning

- a) Fine tuning du modèle 1
  - a. Quelle est la profondeur de l'arbre de décision ?  
On utilisera la méthode `get_depth()`
  - b. On veut maintenant voir comment varie l'accuracy en fonction de l'hyperparamètre `max_depth` de l'arbre de décision
    - i. Créer une liste `max_depth_list` contenant tous les entiers entre 1 et 12 inclus
    - ii. Créer deux listes, `train_accuracy` et `test_accuracy`, qui contiennent les accuracy des arbres de décision entraînés avec les différents `max_depth`.  
C'est-à-dire que `train_accuracy[i]` est l'accuracy de classification du set d'entraînement avec un arbre de décision ayant `max_depth = max_depth_list[i]`
    - iii. Afficher le graph contenant `train_accuracy` et `test_accuracy` en fonction de `max_depth`

On utilisera deux `plt.plot` successifs avant le `plt.show()`

On utilisera l'argument `label` dans les `plt.plot` pour définir le nom de chaque courbe

On montrera les labels sous forme de légende avec `plt.legend()`

iv. Quelle est la meilleure valeur de `max_depth` à choisir ? Pourquoi ?

b) Fine tuning du modèle 2

a. Choisir quelle est la meilleure valeur pour l'hyperparamètre `degree` et pour l'hyperparamètre `kernel`.

c) Validation

a. Télécharger les données de val depuis

[https://drive.google.com/file/d/1MRqAYBPrkg4SFe-YqYOo7Ly1v0CQpwPM/view?usp=drive\\_link](https://drive.google.com/file/d/1MRqAYBPrkg4SFe-YqYOo7Ly1v0CQpwPM/view?usp=drive_link)

b. En reprenant les étapes d'homogénéisation et de preprocessing des images, créer des array numpy `val_images` et `val_labels` contenant respectivement les images et les labels des données de validation.

c. En utilisant le modèle 1, avec la meilleure valeur de `max_depth` définie précédemment, calculer l'accuracy de classification des données de validation.

d. Que peut-on dire de cette valeur ?

e. Comment peut-on l'expliquer ?

d) Augmentation de données. Le but est ici de diversifier nos données d'entraînement en appliquant des transformations aux données disponibles. Pour cela, on créera une méthode `peuplate_and_augment_images_and_labels_lists`, qui remplacera l'existante `peuplate_images_and_labels_lists`. Cette méthode ajoute après l'appel à `cv2.resize` :

a. Une transformation de cropping

```
cropped_image = resized_image[48 :162, 48, 162]
```

b. Une transformation en noir et blanc

```
grey_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
```

```
grey_image = cv2.cvtColor(grey_image, cv2.COLOR_GRAY2BGR)
```

c. Quelle est la dimension d'une `grey_image` après la première ligne ? Quelle est la dimension d'une `grey_image` après la deuxième ligne ? Sur quel paramètre joue-t-on ?

d. Quel est l'intérêt de cette deuxième ligne dans la transformation en noir et blanc ? Dans la liste finale, pour chaque image du dataset initial, on a 3 images : l'image non transformée, l'image avec cropping, l'image en noir et blanc. On n'oubliera pas que toutes les images doivent avoir les mêmes dimensions (=> `cv2.resize`)

e. Entraîner le modèle 1 avec ces nouvelles données (on n'oubliera pas de faire le preprocessing de ces nouvelles images).

f. Comment a évolué l'accuracy de classification sur les données de validation ?

e) Plus de fine-tuning ! Modifier les hyperparamètres des modèles, la taille des images et les transformations de l'augmentation des images pour améliorer l'accuracy de la classification des images test et validation.

f) Autres modèles. Augmenter l'accuracy de classification des images test et validation, en essayant d'autres modèles de classification et en les optimisant.