

Compte rendu du travail fait sur le projet : Semaine 4

Corentin :	2
Pourquoi convertir en HSV ?	2
Comment se fait la conversion ?	2
Conversion RGB to GRAY ?	3
Jami :	3
Arno :	4
Julien :	7
Eliaz	13

Corentin :

Pourquoi convertir en HSV ?

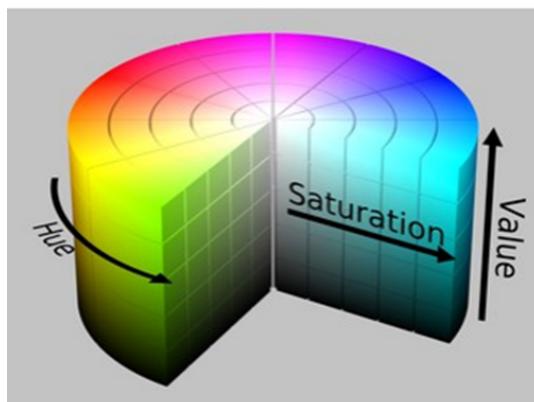
Le HSV est un modèle de couleur qui représente la teinte, saturation et la luminosité de l'image. En R, G, B, nous ne pouvons pas séparer la couleur de l'information de luminance. HSV est utilisé pour séparer l'image de luminance de la couleur de l'information. En vision par ordinateur, on souhaite séparer les composants couleur de l'intensité pour différentes raisons, comme les changements d'éclairage ou la suppression des ombres.

Comment se fait la conversion ?

$$t = \begin{cases} 0, & \text{si } \max = \min \\ (60^\circ \times \frac{g-b}{\max - \min} + 360^\circ) \bmod 360^\circ, & \text{si } \max = r \\ 60^\circ \times \frac{b-r}{\max - \min} + 120^\circ, & \text{si } \max = g \\ 60^\circ \times \frac{r-g}{\max - \min} + 240^\circ, & \text{si } \max = b \end{cases}$$
$$s = \begin{cases} 0, & \text{si } \max = 0 \\ 1 - \frac{\min}{\max}, & \text{sinon} \end{cases}$$
$$v = \max$$

max = la composante r/g/b la plus grande

min = la composante r/g/b la plus faible



0° ou 360° : rouge; 60° : jaune; 120° : vert; 180° : cyan; 240° : bleu; 300° : magenta.

La teinte (Hue) se lit sur un cercle : elle est codée par un angle entre 0 et 360°.

La saturation se lit sur le rayon du cercle : elle est codée par un nombre entre 0 et 1. (de blanc vers la couleur)

La luminosité (Value) se lit sur la coordonnée verticale : tout comme la saturation, elle s'exprime par un nombre entre 0 et 1. (de noir vers la couleur)

Conversion RGB to GRAY ?

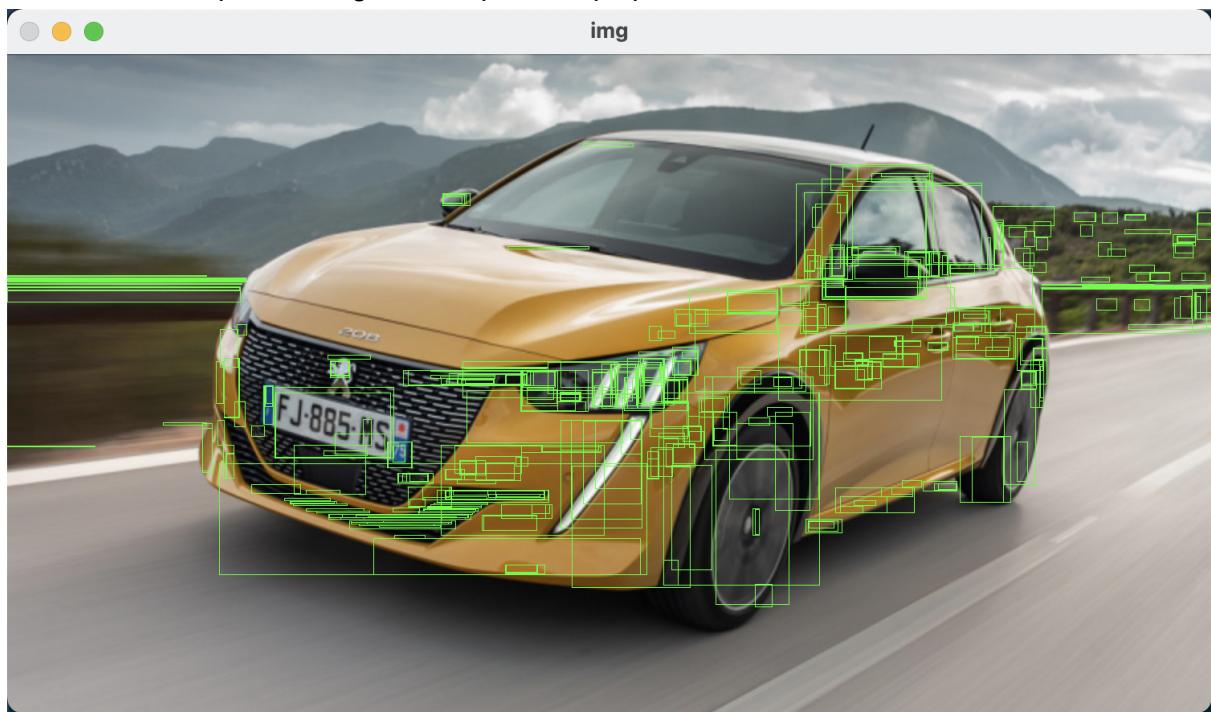
On utilise la méthode pondérée, elle pondère le rouge, vert et bleu en fonction de leurs longueurs d'onde. Niveau de gris = valeur*R+valeur*G+valeur*B
opencv utilise ces valeurs : $0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$

La moyenne pondérée consiste à donner aux valeurs un poids différent, en fonction des divers critères qui rendent compte de l'importance relative de chacun des éléments.

Jami :

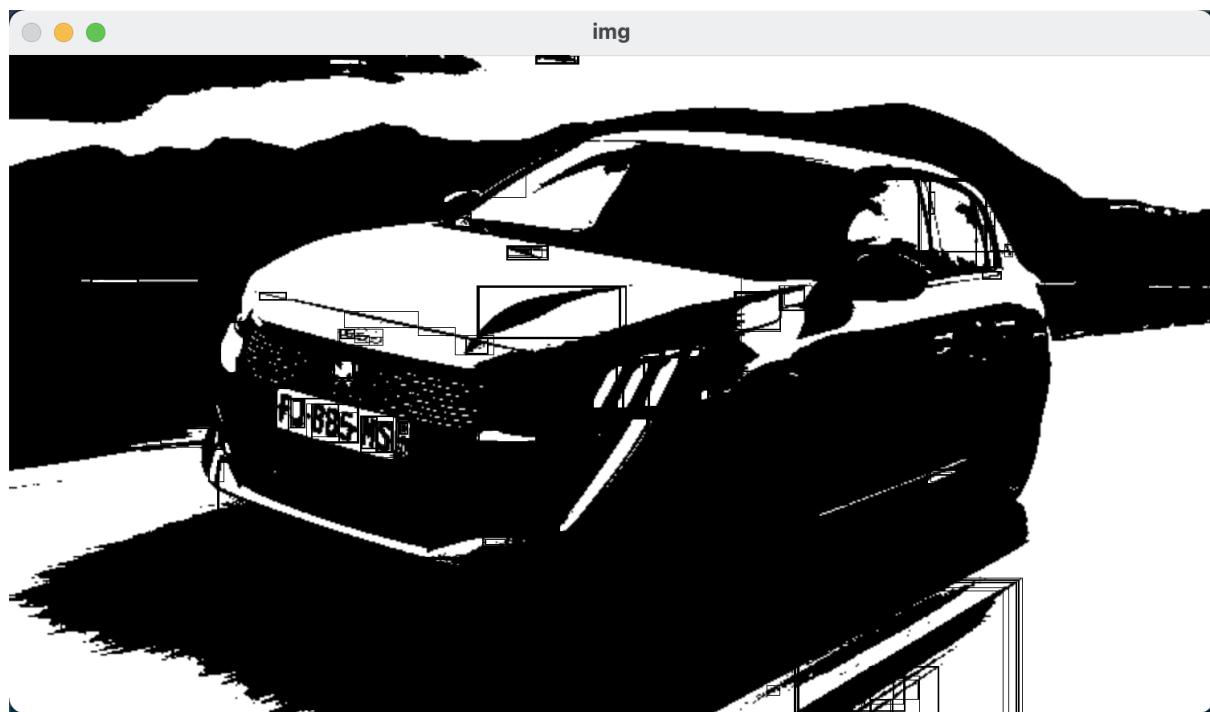
Approfondissement de la méthode vu la semaine dernière “Décomposition/fusion” de manière moins théorique.

Suite à la création d'un algorithme réalisant la méthode de décomposition/fusion on obtient une liste de régions sous forme de coordonnées. Le problème est qu'on perd beaucoup d'éléments dès qu'une image est un peu compliquée.



Cette technique de segmentation est assez peu efficace en l'état avec une image en couleur elle détecte trop de petites zones. Sur le résultat ci-contre, seulement les zones de taille supérieure à la moyenne de toutes les zones ne sont pas gardés, cela permet de trier un minimum les zones segmentés.

En revanche avec une image en Treshold, les résultats sont plus concluants.



Il y a une petite amélioration mais comme le réglage du threshold est manuel, il n'est pas parfait. Je pense donc qu'il faudrait une fonction de Threshold adaptative automatique. Nous avons quand même décidé de ne pas garder cette méthode.

Arno :

Cette semaine, j'ai précisé / affiné les contours. Je commençais d'ors et déjà à avoir les contours de l'objet, cependant je n'avais pas la silhouette, ce qui était le but final. Je n'avais que des contours qui étaient "à l'intérieur" de l'objet. Avec un peu de modification dans les paramètres de la fonction threshold, j'ai pu obtenir ce résultat :



On voit que les contours prennent bien l'ensemble de la voiture. Pour l'instant le résultat est assez satisfaisant. Cependant, le contour représentant l'ombre de la voiture est encore de trop. Pourquoi ? Car lors de threshold, les nuances de gris sont devenues complètement noires.



Voici le résultat obtenu par la fonction threshold. On peut désormais comprendre pourquoi le contours est si bas. Il me faut désormais savoir comment ne pas "récupérer" cette ombre, ou du moins la réduire. Moins l'ombre sera prononcée, plus je m'approcherai de la carrosserie de la voiture et plus les contours seront précis.



Cette fois-ci on voit que la part d'ombre est bien moins importante. Cependant, on a plus l'entièreté de la voiture. La but pour la semaine 5 sera d'évaluer un juste milieu afin d'avoir le plus de détails possible sur l'objet et de réduire les ombres comme dans ce cas ci.

En essayant cette fois-ci sur une voiture entièrement noire. Il est légèrement plus complexe d'avoir des contours aussi précis. On arrive tout de même à obtenir une majorité des contours mais pas l'entièreté. Ça sera donc sur ça que je vais travailler lors de la semaine 5.



De plus avec le groupe segmentation, il va nous falloir une fonction adaptative au threshold en fonction de l'image. Le but sera de faire une fonction prenant une image en paramètre et retourne la même image mais avec le bon niveau de threshold. Prenons l'exemple des photos précédentes. Si on a une image avec une voiture blanche / grise, un certain paramètre de la fonction threshold sera à définir à une certaine valeur. Cependant, si on prend l'image avec la voiture noire, on aura une nouvelle valeur à passer en paramètre dans la fonction threshold. L'objectif est que la fonction threshold s'adapte à toutes les images qui lui seront passées en paramètres et qu'elle puisse retourner le meilleur threshold possible afin de dessiner les contours les plus précis. Grâce à ces résultats la partie classification aura une plus grande facilité à traiter l'image.

Julien :

Regressions linéaire avec Sklearn

- Pour générer les mêmes données, on utilise :
`mp.Random.seed()`

↳ rend les nombres aléatoires prévisibles

- On choisit le nombre de points/samples :

$$m = mb \text{ pts}$$

- On créer un tableau X en 2 dimensions :

`X = mp.Pinspace(d, a, m).reshape(m, mb colonne)`

↳ permet d'obtenir un tableau 1D aplati
d'une valeur `d` à une valeur `a`.

↳ permet de remodeller le tableau, dans
ceux cas on veut un tableau 2D donc
on fera `.reshape(m, 1)`

- on créer un tableau y en 2 dimensions :

`y = X + mp.Random.Random(m, 1)`

↳ créer un tableau de forme spécifiée
(`m, 1`) et le remplit de valeurs aléatoires

- On trace le nuage de points de notre dataset :

`ppt.scatter(X, y)`



- On va importer la classe **Linear Regression** depuis le sub-package **Linear Regression**:

`from sklearn.linear_model import LinearRegression`

- On initialise le modèle :

`model = LinearRegression()`

- On entraîne notre modèle :

`model.fit(X, y)`

- On évalue le modèle :

`model.score(X, y)`

`0.88742`

- On fait une prediction :

`predictions = model.predict(X)`

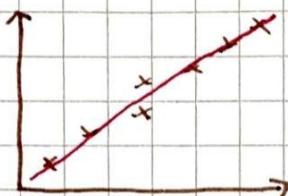
Regressions Linéaire avec Sklearn

- On affiche ensuite les résultats de la prediction :

plt.scatter(X, y)

plt.plot(X, prediction, c='r')

↳ dessiner la droite des résultats de la prediction



Dataset Boston housing data

- On importe les librairies nécessaires :

```
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
%matplotlib inline
```

Importation et compréhension des données:

- On importe le dataset

```
from sklearn.datasets import  
load_boston
```

```
donnees_boston = load_boston()
```

- On peut visualiser ce que contient la librairie :

```
donnees_boston.keys()
```

↳ permet de lister les propriétés

Dans notre cas on voit :

- **data** : infos sur les maisons
- **target** : le prix des maisons
- **feature_names** : nom des caractéristiques du dataset
- **DESCR** : description du dataset

- On peut afficher la description du dataset des données - boston. DESCR.split ("Im")
- split("Im") : diviser par Pigme
- équivaut à splitPimes()

A partir de la description on peut connaître le nombre d'instances et le nombre d'attributs

Préparation des données :

- On transforme notre dataset en Data Frame grâce à pandas :

data_df = pd.DataFrame (dommes-boston.data, columns = dommes-boston.features - names)

- On peut regarder à quoi ressemble notre Data Frame :

data_df.head()

↳ affiche les 5 premières lignes

Dataset Boston housing data

- on crée une nouvelle colonne "PRIX"
`data_df['PRIX'] = donnees_boston.target`
PRIX correspond aux targets
- on vérifie s'il n'y a pas de valeurs nulles :
`data_df.isnull().sum()`

Création du modèle :

- Avant de créer notre modèle, on va choisir quelle variable a une forte relation linéaire avec la variable "PRIX".

Pour choisir, on va créer une matrice de corrélation :

Les coefs de corrélation se situent dans l'intervalle [-1; 1]

- proche de 1 : forte corrélation positive
- proche de -1 : forte corrélation négative
- proche de 0 : faible corrélation

Eliaz

Cette semaine j'ai mis en pratique ce que j'avais fait en théorie la semaine dernière. Ici le dataset pour lequel j'ai créé un réseau de neurones est le `load_digits` dataset de `sklearn` qui comporte 1797 images de 8x8 pixels avec un chiffre dessus. Ce dataset contient également quel est ce chiffre (le label).

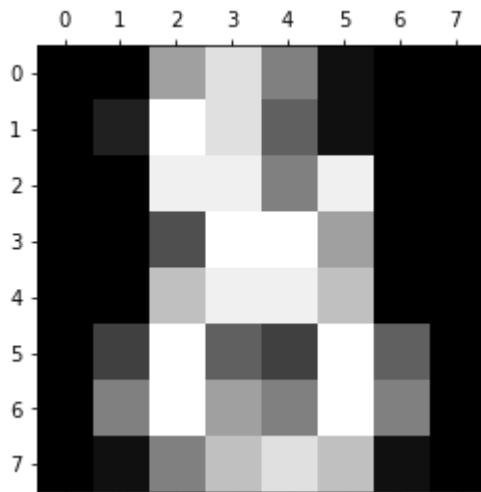


Figure ci-dessus : image n°1796 du dataset, il s'agit ici d'un 8.

Je choisis ici de diviser ce dataset en deux car il comporte peu d'entrées et donc relativement peu de paramètres.

En effet, le réseau de neurones que j'ai choisi de créer à ici $8 \times 8 = 64$ nodes en entrée, 2 couches de 16 nodes en couche cachée (nombre choisi de façon arbitraire) et 10 nodes sur sa couche de sortie. Cela fait un total de $64 \times 16 + 16 \times 16 + 16 \times 10 = 1440$ poids et $16 + 16 + 10 = 42$ biais, soit un total de 1482 paramètres différents.

Pour l'entraînement de ce réseau de neurones, `Sklearn` approxime les valeurs des paramètres en essayant de trouver les valeurs qu'ils doivent prendre pour avoir la meilleure exactitude. Cela revient ici un peu à trouver le minimum d'une fonction avec 1482 paramètres.

Ici l'entraînement est paramétré pour se terminer quand le dataset d'entraînement est épuisé ou que l'entraînement ne s'améliore plus et que les paramètres optimaux ont donc été trouvés.

L'évaluation du progrès de l'entraînement se fait en regardant la valeur de la fonction de coût (similaire à la fonction perte, `loss` en anglais). Dans `Sklearn`, l'attribut qui correspond à cette fonction pour examiner le progrès s'appelle `.loss_curve_`. Elle se calcule en faisant :

$$\Sigma((\text{résultat observé} - \text{résultat attendu})^2)$$

ce qui permet d'avoir une fonction qui est grande quand le résultat n'est pas celui attendu et petite quand il correspond.

$$\left. \begin{array}{l} 0.0016 \leftarrow (0.04 - 0.00)^2 + \\ 0.0014 \leftarrow (0.04 - 0.00)^2 + \\ 0.0046 \leftarrow (0.07 - 0.00)^2 + \\ 0.0011 \leftarrow (0.97 - 1.00)^2 + \\ 0.0102 \leftarrow (0.10 - 0.00)^2 + \\ 0.0004 \leftarrow (0.02 - 0.00)^2 + \\ 0.0051 \leftarrow (0.07 - 0.00)^2 + \\ 0.0082 \leftarrow (0.09 - 0.00)^2 + \\ 0.0149 \leftarrow (0.12 - 0.00)^2 + \\ 0.0044 \leftarrow (0.07 - 0.00)^2 \end{array} \right\} 0.05$$

Figure ci-dessus : Exemple de calcul du coût pour une instance où le résultat attendu est 3.

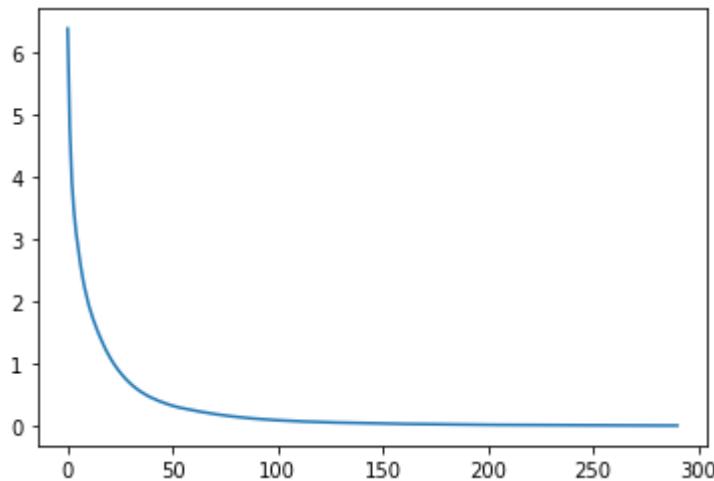


Figure ci-dessus : Valeur des "loss" (voir explication au dessus) en fonction des itérations de l'entraînement.

On observe qu'avec l'entraînement, la fonction loss se stabilise à un minimum, après cela, il n'est plus nécessaire de continuer l'entraînement car les paramètres du réseau de neurone ne changent plus : la fonction avec comme paramètres ceux du réseau de neurone à atteint son minimum.

Ce réseau de neurones permet ici d'avoir un taux de prédiction autour des 95%, ce qui est similaire, voire supérieur à ce dont est capable un humain avec ces images en résolution très basse.