# Informatique

# Intelligence artificielle

***Réseau de neurones***
***Convolution d'images – Détection de contours***

# *Contexte industriel*

Dans de nombreux systèmes industriels, robots de tri, véhicules autonomes, contrôle qualité en ligne de production, la chaîne d'information repose sur une étape cruciale : l'analyse d'images en temps réel. Un capteur (caméra) acquiert une image, un algorithme la traite pour en extraire une information pertinente (présence d'un défaut, détection d'un obstacle, reconnaissance d'un objet), puis un actionneur réagit en conséquence.

Ce sujet s'inscrit dans cette logique. Vous allez construire pas à pas une chaîne de traitement capable de détecter les contours d'une image, opération fondamentale en vision industrielle :

- La convolution par noyau Laplacien joue le rôle d'un filtre de traitement du signal, mettant en évidence les variations brutales de luminosité, c'est-à-dire les contours.
- La binarisation transforme ce signal continu en une information binaire (contour / pas de contour), comme un comparateur dans une chaîne d'acquisition.
- Enfin, un réseau de neurones est entraîné à reproduire automatiquement cette chaîne de traitement, illustrant comment un système peut apprendre une loi de comportement à partir de données. C'est le principe au cœur de l'intelligence artificielle embarquée dans les systèmes modernes.

# *La convolution appliquée aux images*

Dans le programme d'ITC, est écrit :

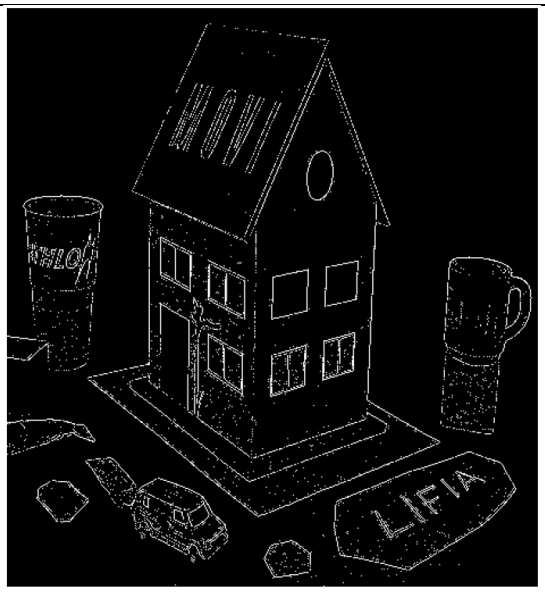
| | |
|---|---|
| Matrices de pixels et images. | Algorithmes de rotation, de réduction ou d'agrandissement. Modification d'une image par convolution : flou, détection de contour, etc.<br>*Les images servent de support à la présentation de manipulations de tableaux à deux dimensions.* |

L'application de la convolution sur une image consiste à créer une nouvelle image dont les valeurs des pixels sont calculées, par chacun, en tenant compte de ses voisins et d'un noyau. Ci-dessous, un extrait du résumé du cours de première année :

<table>
<tr><td colspan="3">Convolution</td></tr>
<tr><td>Principe avec un noyau de dimensions 3x3</td><td colspan="2">
Noyau $K = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix}$ à $n$ lignes et $n$ colonnes<br>
$Im = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} \end{bmatrix}$ ; $P_{ij} = [R_{ij}, G_{ij}, B_{ij}]$<br>
$Im' = \begin{bmatrix} P'_{11} & P'_{12} & P'_{13} & P'_{14} & P'_{15} \\ P'_{21} & P'_{22} & P'_{23} & P'_{24} & P'_{25} \\ P'_{31} & P'_{32} & P'_{33} & P'_{34} & P'_{35} \\ P'_{41} & P'_{42} & P'_{43} & P'_{44} & P'_{45} \\ P'_{51} & P'_{52} & P'_{53} & P'_{54} & P'_{55} \end{bmatrix}$ ; $P'_{ij} = [R'_{ij}, G'_{ij}, B'_{ij}]$ Intérieurs<br>
$P'_{ij} = K * M_{ij}$ ; $M_{ij} = \begin{bmatrix} P_{i-1,j-1} & P_{i-1,j} & P_{i-1,j+1} \\ P_{i,j-1} & P_{i,j} & P_{i,j+1} \\ P_{i+1,j-1} & P_{i+1,j} & P_{i+1,j+1} \end{bmatrix}$<br>
$P'_{ij} = K_{11}P_{i-1,j-1} + K_{12}P_{i-1,j} + K_{13}P_{i-1,j+1} + K_{21}P_{i,j-1} + K_{22}P_{i,j} + K_{23}P_{i,j+1} + K_{31}P_{i+1,j-1} + K_{32}P_{i+1,j} + K_{33}P_{i+1,j+1}$<br>
$K_{ij}P_{ij} = K_{ij}[R_{ij}, G_{ij}, B_{ij}] = [K_{ij}R_{ij}, K_{ij}G_{ij}, K_{ij}B_{ij}]$
</td></tr>
<tr><td>Laplacien<br>Contours</td><td>$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$</td><td>  </td></tr>
<tr><td>Gestion des bords</td><td colspan="2">On peut les laisser à leur valeur d'origine, redimensionner l'image de sortie en les enlevant, ou traiter les formules au cas par cas</td></tr>
<tr><td>Normalisation</td><td colspan="2">Afin de maintenir une moyenne des RGB par convolution, on peut normaliser la matrice en divisant chacun de ses termes par la somme de tous ses termes</td></tr>
<tr><td>Re limitation des résultats</td><td colspan="2">Dans le cadre des BMP par exemple et pour éviter l'overflow, on relimite les valeurs de R, G et B afin de les maintenir dans l'intervalle [0,255] avec une formule comme : $E = min(max(E, 0),255)$</td></tr>
</table>

# *Contexte de cette étude*

Notre objectif est de créer un réseau de neurones capable de transformer une image en couleur en une image contenant ses contours en noir et blanc. On utilisera un noyau Laplacien de dimension 3x3.


La première approche consiste à voir les choses de manière **globale** :

- Entrées du réseau : Liste de toutes les valeurs RGB de tous les pixels de l'image
- Sortie du réseau : Image en noir et blanc contenant les contours

Cette approche nécessiterait un entraînement gigantesque sur de grandes images… Approche abandonnée.

La seconde approche mise en œuvre dans la suite de ce sujet consiste à réaliser une **approche locale**. Toujours pour simplifier l'apprentissage (du réseau de neurones Python, et de celui du lecteur), on considèrera que les images ont été passées en nuances de gris (apprentissage de 9 valeurs en entrée plutôt que 27). On apprendra alors au réseau de neurones à appliquer automatiquement à chaque pixel les transformations suivantes :

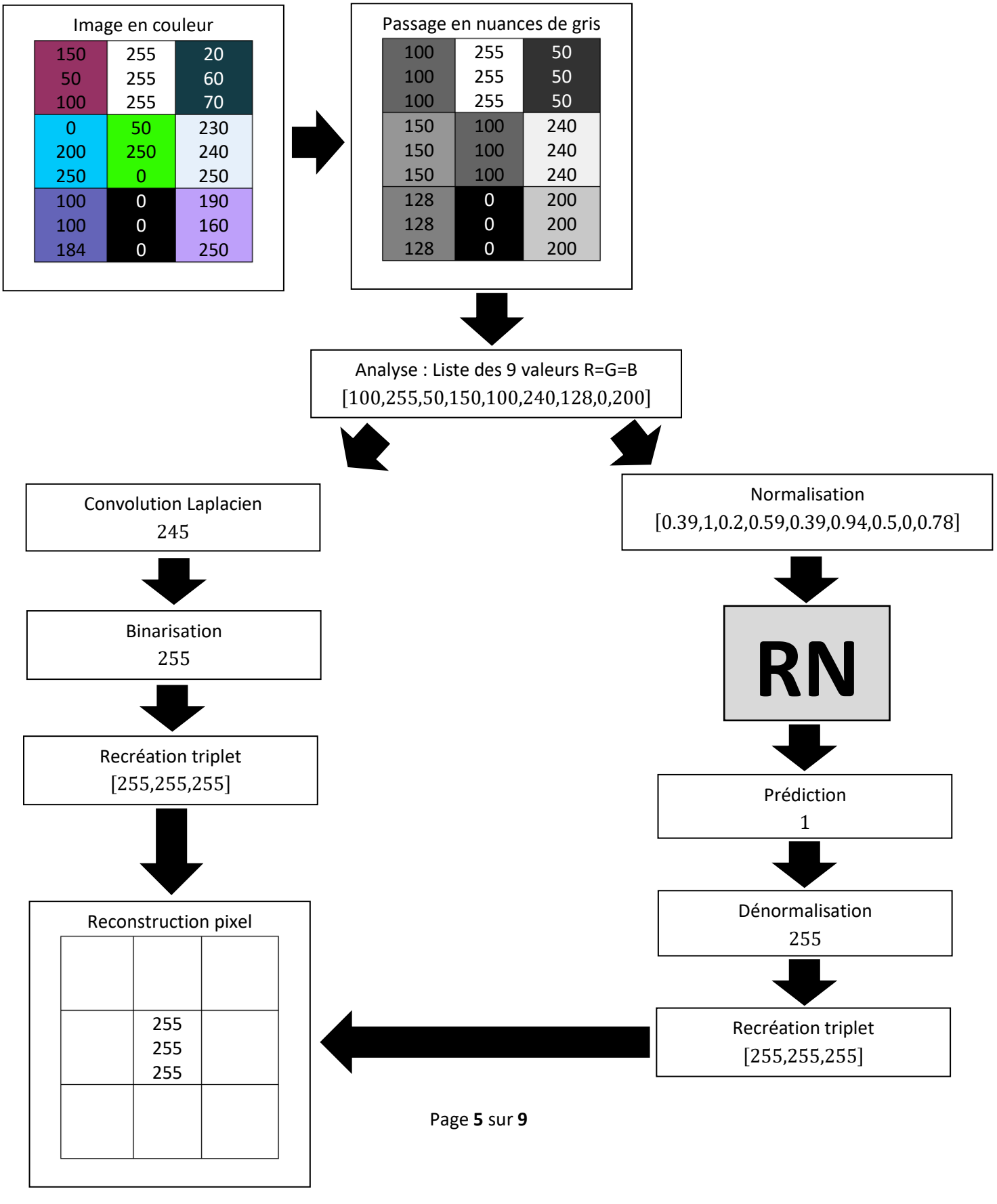
- Convolution classique
- Binarisation (noir ou blanc)

En fin de sujet, nous étendrons l'analyse aux 27 valeurs des images en couleurs.

Les données seront donc les suivantes :

- Entrée du réseau : 9 valeurs R=G=B de chacun des 9 pixels de l'image en nuances de gris autour d'un pixel
- Sortie du réseau : Valeur R=G=B valant 1 ou 0 définissant la présence d'un contour ou non en ce point

Illustration de la procédure :



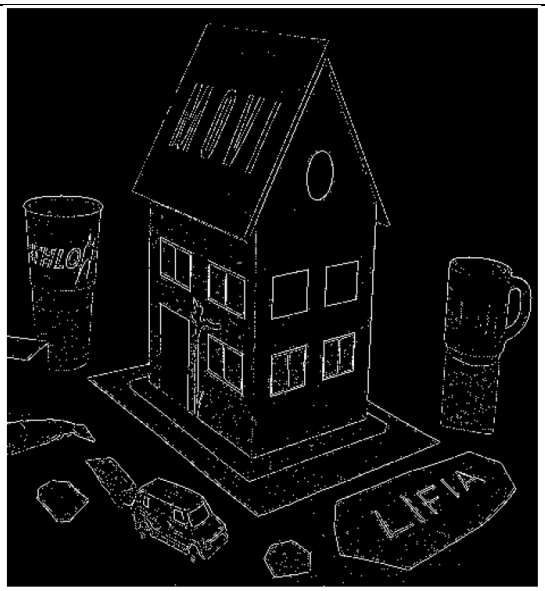
# *Code élèves*

Afin d'assurer un fonctionnement rapide sur tous les ordinateurs, je vous mets à disposition un dossier élèves.

Exécutez le code fourni, il réalisera les travaux suivants :

- Ouverture et affichage sur la figure 1 de l'image fournie « Image.bmp »
- Transformation de l'image en nuances de gris (R=G=B pour chaque pixel) et affichage du résultat sur la figure 2
- Application du noyau de convolution Laplacien à l'image en nuances de gris et affichage du résultat sur la figure 3
- Application d'une binarisation (transformation « noir et blanc » avec un seuil S) afin d'obtenir une image ne contenant que des pixels blancs (R=G=B=255) ou noirs (R=G=B=0) et affichage du résultat sur la figure 4
- Application d'une fonction de convolution locale faisant le même travail que toutes les fonctions précédentes sur l'image en nuances de gris et affichage du résultat sur la figure 10 (résultat identique à celui de la figure 4)

Prenez le temps de comprendre le code élève, et en particulier, les fonctions f_Convolution et f_Conv_loc_NB.

# *Création des datasets*

Pour entraîner le réseau de neurones, nous allons lui fournir deux listes :

- Donnees_Sources : Liste de listes de 9 valeurs normalisées (entre 0 et 1) des R de 9 pixels en nuances de gris (image 3x3 générée aléatoirement)
- Données_Cibles : Résulat normalisé (entre 0 et 1) de la convolution et binarisation

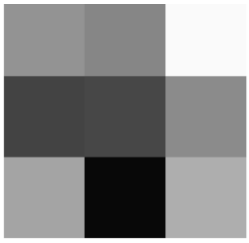
On rappelle que le module random possède la fonction randint(a,b) renvoyant un entier compris entre a et b inclus.

Pour générer une image de dimensions nxn noire de toute pièce dans le bon format, on pourra écrire :

```
im = np.zeros((n,n,3),dtype='uint8')
```

**Question 1: Proposer une fonction f_Aleat_NG() renvoyant un array représentant une image 3x3 en nuances de gris**

Vérifier par exemple : `f_Affiche(20,f_Aleat_NG())`



**Question 2: Proposer une fonction f_Analyse(imng) prenant en argument une image en nuances de gris, et renvoyant la liste des valeurs R de tous ses pixels ligne par ligne**
Astuce : utiliser float() pour faire disparaître np.float64

Vérifier par exemple :

```
>>> f_Analyse(f_Aleat_NG())
[0.25882352941176473, 0.6745098039215687, 0.33725490196078434, 0.611764705882353, 0.607843137254
9019, 0.47058823529411764, 0.8705882352941177, 0.7294117647058823, 0.7843137254901961]
```

**Question 3: Proposer une fonction f_creation_donnees(N) renvoyant les deux listes Sources et Cibles attendues pour N images aléatoires de 9 pixels en nuances de gris**
Vérifier par exemple :

```
>>> f_creation_donnees(2)
([[0.19215686274509805, 0.3843137254901961, 0.6039215686274509, 0.2901960784313726, 0.2784313725
490196, 0.6901960784313725, 0.8470588235294118, 0.7215686274509804, 0.9725490196078431], [0.2039
2156862745098, 0.5568627450980392, 0.17254901960784313, 0.09019607843137255, 0.4627450980392157,
0.8, 0.8156862745098039, 0.9176470588235294, 0.36862745098039222]], [1.0, 1.0])
```

# *Apprentissage*

Nous allons utiliser un MLPClassifier « Multi-Layer Perceptron » qui classifie les données. La sortie vaut soit 0, soit 1.
Si besoin, pour installer le module, tenter dans la console « `pip install scikit-learn` » ou « `python -m pip install scikit-learn` ».
Quelques rappels :

| | |
|---|---|
| Découpage des données normalisées (sources et cibles) en set d'entraînement et de test | `from sklearn.model_selection import train_test_split`<br>`x_train,x_test,y_train,y_test = train_test_split(x,y)` |
| Import et création du réseau de neurones multicouches « Classifier » | `from sklearn.neural_network import MLPClassifier`<br>`rn = MLPClassifier(solver='sgd')` |
| Entrainement du modèle | `rn.fit(x_train,y_train)` |
| Prédiction<br>`Donnee` : Liste des R normalisés | `rn.predict([Donnee])[0]` |
| Score sur données test | `rn.score(x_test,y_test)` |
| Sauvegarde de modèle | `from joblib import dump`<br>`dump(rn,'rn.joblib')` |
| Chargement de modèle | `from joblib import load`<br>`rn = load('rn.joblib')` |

Vous allez maintenant écrire les lignes de code permettant de séparer les données en un set d'entrainement et un set de test, de créer le réseau de neurones, de l'entraîner, et d'afficher son pourcentage de réussite dans la console. Vous choisirez N pour une réussite supérieure à 99%.

**Question 5: Proposer le code attendu**

# *Application*

**Question 6: Proposer une fonction f_Conv_loc_rn(Mng,_) réalisant une convolution locale par réseau de neurones et renvoyant donc une liste [R,G,B] où R=G=B valant soit 0, soit 255, selon la prédiction du réseau**
**Question 7: Tester finalement la convolution par réseau de neurone**

# *Pour aller plus loin*

Maintenant que vous avez tout compris sur une donnée d'entrée simple de 9 valeurs des pixels voisins en nuances de gris, reprenez votre code pour qu'il réalise le même travail en prenant en entrée les 27 valeurs des pixels voisins sur l'image en couleurs. Consultez en page suivante un schéma de la démarche améliorée attendue.

Les données seront donc les suivantes :

- Entrée du réseau : 27 valeurs R,G,B de chacun des 9 pixels de l'image en couleur autour d'un pixel
- Sortie du réseau : Valeur R=G=B valant 1 ou 0 définissant la présence d'un contour ou non en ce point

Illustration de la procédure :