# PROJET : Machine Learning

## Classification automatique supervisée d'images naturelles et artificielles

L'objectif est de concevoir un système capable de classer automatiquement des images en différentes catégories. On dispose pour cela d'une banque de 2288 images numérotées. Chaque image possède 2 catégories principales et chaque catégorie possède plusieurs sous-catégories.

La première catégorie représente la classe principale de l'image, elle peut être :

- Artificielle (constructions humaines)
- Naturelle (image de nature)

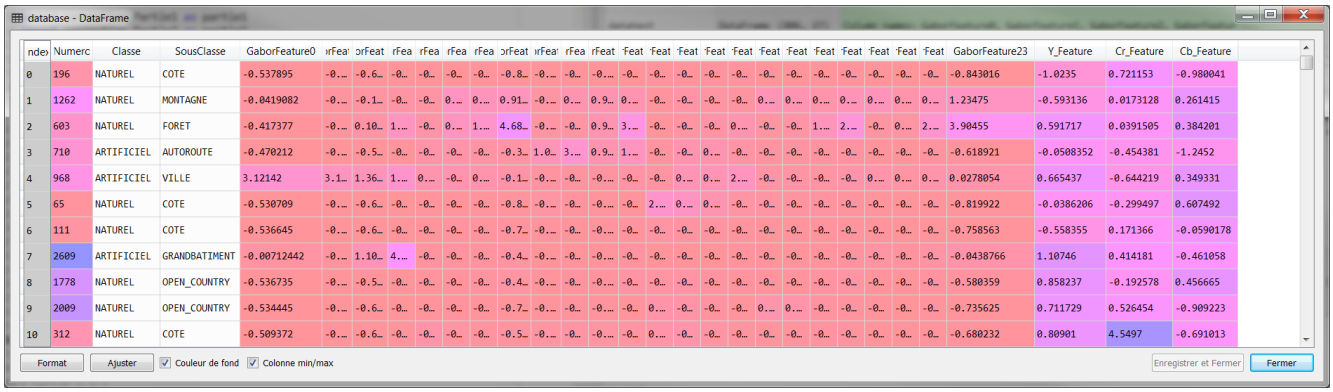
La seconde catégorie représente la sous-classe de l'image, il y en a 8 :

- Côte (plage, mer…)
- Forêt
- Autoroute
- Ville (Photo prise de loin de l'ensemble de la ville)
- Montagne
- Paysage ouvert (landes, désert…)
- Rue
- Grand bâtiment

Chaque image est décrite à l'aide de 27 vecteurs de caractéristiques (features en anglais). Les 24 premiers vecteurs de caractéristiques sont issus d'un banc de filtres de Gabor et consistent à récupérer l'énergie dans chacune des bandes de fréquences (ici 24). Les 3 derniers vecteurs de caractéristiques sont issus de la luminance Y et de la chrominance Cb puis Cr.

Pour information, les images sont accessibles dans le fichier images.zip et peuvent être visualisées.

Structure du fichier cvs, ici visualisé dans une variable panda (dataframe):



# Première partie : Machine learning classique

L'objectif de ce projet est de proposer un classifieur parmi un SVM et un MLP qui permette de répondre le mieux au problème de classification posé (classification images naturelles/artificielles, classification par catégories d'images).
Pour cela, commencer par partitionner la base de données en deux bases : apprentissage et test.
Visualiser les données de la base de test en réalisant une ACP et une ALD et en projetant les données sur les 2 premiers axes.
Ensuite, pour les 2 classifieurs (MLP et SVM), choisir la structure la plus adaptée au problème de classification. On utilisera comme critère de sélection une métrique calculée par validation croisée sur la base d'apprentissage.
Une fois la structure des classifieurs choisie, évaluer et comparer leurs performances sur la base de test, en prenant soin de calculer les intervalles de confiance des métriques de performances choisies.
Conclure sur le choix des classifieurs retenus.

Vous pourrez évaluer la sensibilité des résultats à la taille de la base d'apprentissage. Vous pourrez aussi évaluer la perte en performances quand on supprime les 3 dernières caractéristiques, c'est-à-dire la luminance Y, la chrominance Cb et Cr.

Avant de programmer un MLP ou un SVM, il est nécessaire d'étudier sur le site de scikitl-earn la documentation relative à ces classifieurs : MLPClassifier et svc, ainsi que la fonction permettant la validation croisée, cross_validate. Il est nécessaire de prendre le temps de comprendre les paramètres d'entrées et les attributs associés à chaque fonction.

https://scikit-learn.org/stable/modules/neural_networks_supervised.html#multi-layer-perceptron
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/modules/cross_validation.html
**https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate**

Pour le MLP, on fixera le nombre de couches cachées à 1. On optimisera le nombre de neurones dans la couche cachée et on choisira la fonction d'activation associée aux neurones.
Pour le SVM, on choisira le type de noyau, la constante de pénalité C et la stratégie choisie pour obtenir un classifieur à M quand M est supérieur à 2.

Un fichier d'aide pour le démarrage est donné sur Chamilo, mais il n'est pas obligatoire de l'utiliser.

# Deuxième partie : Deep Learning — CNN (Convolutional Neural Networks)

***Important:*** *Avant de commencer cette partie, vérifier que votre ordinateur possède la bibliothèque keras en rentrant dans la console : import keras. Ainsi que la bibliothèque PIL en rentrant dans la console : import PIL.*

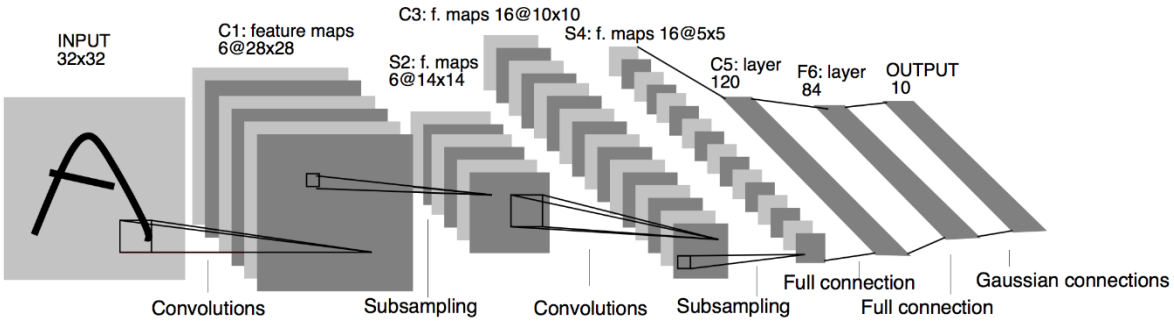
Le Deep Learning (ou Apprentissage profond) est un ensemble de méthodes d'apprentissage automatique appartenant au Machine Learning. L'une d'entre elles consiste à mettre en entrée d'un MLP directement les images qui seront traitées et décomposées par un ensemble de filtres de convolution. Cette méthode représente le CNN (voir figure représentant LeNet)
Ils existent de nombreuses bibliothèques permettant de générer un tel réseau sous Python. La bibliothèque que vous utiliserez sera keras qui est actuellement en plein essor.
**https://keras.io/getting-started/sequential-model-guide/**

## I. Architecture du CNN et du réseau de neurones

***Objectif*** *: Comprendre la définition d'un premier réseau de neurones sous Keras*

A présent, nous utiliserons le fichier 'database.csv' uniquement dans le but d'acquérir les labels des images puisque le CNN créera de lui-même ses propres caractéristiques.
Nous vous donnons un premier exemple d'apprentissage par réseaux de neurones de type CNN (Convolutionnal Neural Network) basée sur le réseau proposé par Yann Lecun.

L'architecture "LeNet" sur lequel nous allons baser notre modèle a été conçu par Yann Le Cun.




*Architecture LeNet par Yann Le Cun établit en 1998*

Elle est constituée de :

- Une première couche qui inclut : une convolution, une fonction d'activation et un pooling (sous-échantillonnage). Ce sont les caractéristiques créées par le CNN.
- Une seconde couche qui effectue la même chose mais avec plus de filtres pour la convolution. On obtient ainsi plus de paramètres et de détails de l'image.
- L'entrée du réseau MLP, un nombre d'entrées qu'il faudra prendre soin de choisir et qui ne devra pas être trop grand (>2500) au risque d'avoir un temps de calcul trop long. Ce nombre se calcule à partir de la taille d'image sortie du dernier pooling et du nombre de filtres de la dernière convolution.
- La couche cachée des neurones du MLP, malheureusement, il n'y a pas de formule magique pour savoir combien de neurones il faut garder pour cette couche.
- Et enfin, la couche de sortie, dont le nombre de neurones est égal au nombre de classes.

Nous allons reproduire cette architecture avec la particularité, que nos images seront en couleurs et non en niveaux de gris.

## II. L'apprentissage, validation et test

***Objectif** : Réaliser l'apprentissage puis la validation sur plusieurs générations (epoch en anglais). Tester votre modèle sur la base de test.*
***Important:** Attention à bien maîtriser l'architecture de votre CNN et de votre MLP. Les temps de calculs pourraient devenir très longs.*

1. Implémentez votre premier réseau, pour ceci regardez les models que Keras propose. Le modèle Sequential de keras.models, permet de construire son propre réseau (allez voir la documentation) , la fonction add permet de rajouter les couches désirées, ces couches seront bien évidemment des couches de convolution (Conv2D) , mais aussi du pooling (sous-échantillonnage, par exemple MaxPooling2D), éventuellement une couche de Dropout (pour déconnecter un certains nombres de neurones), puis sur la fin la fonction Flatten() pour convertir en vecteur une dimension puis la fonction Dense qui réduit le vecteur de sortie à un nombre désiré. Enfin il faudra choisir un optimizer (calcul de fonction de pertes) puis finalement utiliser la fonction compile pour construire votre réseau, il sera alors prêt à l'apprentissage, que l'on appellera avec la fonction fit()

   Un exemple de construction :

```
def define_model():
        model = Sequential()
        model.add(Conv2D(16,(3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(256, 256, 3)))
        model.add(MaxPooling2D((2, 2)))

        model.add(Conv2D(8,(5, 5), activation='relu', kernel_initializer='he_uniform', padding='same'))
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(0.2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
        model.add(Dropout(0.2))
        model.add(Dense(8, activation='softmax'))
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
        return model
```

2. Etudiez les différents paramètres qui vont être importants pour bien faire fonctionner votre CNN. Par exemple :
   - Augmenter le nombre d'epoch
   - L'augmentation du nombre de filtres. Prenez soin de bien calculer le nombre d'entrées de votre MLP.
   - Des techniques de régularisation comme le Dropout ou le BatchNorm2D (voir la documentation)

- Modifier la taille de vos images en entrée
- Modifier la taille des noyaux de convolution de vos filtres
- Ajouter de nouvelles couches de convolution à votre modèle pour obtenir plus de détails de l'image
- …

3. A cette fin il sera utile de sortir les informations de accuracy et loss pour la base d'apprentissage et la base de validation afin de les tracer en fonction des epochs (fonction history), comment déterminez-vous le bon nombre d'epochs ?
4. Tester sur votre base de test les performances de votre classifieur.
5. Comparer avec les résultats des autres méthodes.

# III. Le transfert learning

***Objectif:*** *Utiliser les résultats des énormes bases de données déjà entrainées dans le monde et disponibles.*

L'idée de l'apprentissage par transfert est donc d'utiliser des réseaux connus formés pour une tâche peu éloignée de celle que nous voulons résoudre et de tirer parti des réseaux déjà formés pour résoudre notre propre problème de classification. Ceci peut être fait de plusieurs manières :

- Utilisation directe des couches convolutionnelles entraînées, comme extracteur de caractéristiques pour votre propre classificateur, puis entraînement du classificateur uniquement (les couches entièrement connectées sont dédiées au processus de classification).
- Utilisez des parties des modèles entraînés pour l'extraction de caractéristiques de bas niveau (en gelant les paramètres correspondants) et réentraînez les couches convolutives proches de la sortie afin de les adapter à votre propre tâche.
- Utilisez les paramètres préformés comme valeurs initiales de votre propre CNN et formez votre CNN à partir de ces valeurs initiales.

L'idée principale de l'apprentissage par transfert est d'accélérer le processus d'apprentissage et de tirer parti des paramètres appris sur de très grands ensembles de données. Par exemple, le réseau VGG16 a été entraîné sur un milliard d'images pour reconnaître des objets dans une image parmi 1000 classes différentes (cf. le jeu de données ImageNet [1]) !

Les trois modèles les plus populaires utilisés pour l'apprentissage par transfert sont les suivants :

- VGG16 ou VGG19 [2]
- GoogLeNet (e.g. InceptionV3)
- Réseau résiduel (par exemple, ResNet50)

Keras donne accès à un certain nombre de modèles pré-entraînés très performants qui ont été développés pour des tâches de reconnaissance d'images et notamment aux trois modèles cités précédemment.

Nous proposons ici d'utiliser un réseau VGG16 pré-entraîné comme extracteur de caractéristiques. Pour ce faire, nous allons télécharger les paramètres du réseau VGG16 et remplacer les couches de classification supérieures du VGG16 conçues pour une tâche de classification de 1000 classes par nos propres couches de classification dédiées à une tâche de classification de 8 classes. L'architecture du réseau VGG 16 est la suivante :

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
```

Pour réaliser ce transfert issu de Vgg16 sur vos données et vos classes, utilisez le code fourni et adaptez le si besoin.

Lors du chargement du VGG16, le paramètre include_top=False signifie que vous voulez supprimer les trois dernières couches du VGG16 (partie classificateur). Ensuite, la boucle avec l'instruction layer.trainable = False signifie que vous ne voulez pas ré-entraîner la couche convolutive. La seule partie du réseau qui va être ré-entraînée sur vos propres données, est la couche entièrement connectée avec 20 neurones et la couche de sortie avec 8 sorties.

## IV Fine-tunning

Pour aller plus loin dans le transfert learning, on peut reprendre l'étape précédente mais au lieu de juste récupérer les caractéristiques finales issues du CNN puis d'apprendre la classification de nos 8 classes, il est possible de "dégeler" quelques couches finales du CNN et de ré-apprendre les poids de ces couches dégelées sur nos données.

Pour ceci il suffit de dire que les couches peuvent être ré-entrainées donc layer.trainable = True, mais surtout ne pas tout refaire et bien choisir le nombre de couches à ré-entrainer.
Soit Nbr_couches_gelees le nombre de couches du modèle qui restent gelées:
Alors une boucle sur les couches après ce nombre permet de les dégeler en appliquant l'instruction layer.trainable=True
Par exemple:

```
for layer in model.layers[Nbr_couches_gelees:]:
    layer.trainable = True
```

Il faut ensuite bien sur re-compiler ce nouveau modèle puis relancer l'apprentissage et enfin tester.

Vous pouvez tester ce transfert learning à partir de Vgg16, ou InceptionV3 par exemple. Regardez bien le modèle du reseau avant de choisir votre nombre de couches.

**[1] *ImageNet*. http://www.image-net.org**
**[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).**

# ANNEXE

**Utils_ClassIm.py**
Cette bibliothèque locale a été créée dans le but de vous simplifier la compréhension de la base de données. Elle permet par exemple de transformer les classes et sous-classes en entier tel que :

- ARTIFICIEL = 0
- NATUREL = 1
- COTE = 2
- FORET = 3
- AUTOROUTE = 4
- VILLE = 5
- MONTAGNE = 6
- OPEN_COUNTRY = 7
- RUE = 8
- GRANDBATIMENT = 9

| **Fonctions** | **Sortie** |
|---|---|
| set_labels (d, col2, col8) | Écris dans la DataFrame aux colonnes correspondantes, le champ des classes en type string. Renvoie une DataFrame. |

| | |
|---|---|
| unset_labels (d, col2, col8) | Écris dans la DataFrame aux colonnes correspondantes, le champ des classes en type int. Renvois une DataFrame. |
| start_time (index) | Enclenche un chrono à l'index correspondant (max 10 chronos) |
| stop_time (index) | Arrête un chrono à l'index correspondant et renvoie. Renvoie un float. |
| show_database (data, col2, col8) : | Affiche le nombre d'images par classe pour une DataFrame donnée avec les colonnes correspondantes aux classes. |
| shake_database (path) | Mélange les lignes d'un fichier csv fourni. |
| lire_images(root_dir, path_cvs ,num, sous_ech=2) | Permet de lire les images de la base, de les sous-échantilloner si besoin et de les sortir dans un tableau T ainsi que leurs labels dans C1 et C2 |
| lire_database(path, num) | Permet de récupérer le numéro d'une image et ses classes pour une valeur num de la base csv. |

## Pandas

Pandas permet de manipuler de grands volumes de données structurées de manière simple et intuitive. Pandas emploie le type "DataFrame" pour stocker les matrices, avec de nombreux outils pour modifier et transformer les données.

| **Définitions** | **Ligne de commande** |
|---|---|
| Importer la bibliothèque | import pandas as pd |
| Lire un fichier *. csv et le stocker dans une variable | data = pd .read_csv ("./file.csv") |
| Sauvegarder une DataFrame en fichier *. csv | data.to _csv ("./file.csv") |
| Créer un tableau de type DataFrame | pd.DataFrame (tableau) |
| Accéder à une partie d'une DataFrame | dataslice = data.iloc [li,col] |
| Concaténer 2 DataFrame | pd.concat ((data1, data2), axis=1) |
| Convertir le type des données d'une DataFrame (en int, str, float…) | data = dataframe. astype (type) |

## Validation croisée

La validation croisée permet de mesurer la fiabilité de notre apprentissage. Lorsque l'on change les paramètres d'un classifieur, il est important de ne pas faire en sorte de chercher à améliorer le score sur la base de test, cela atténuerait les performances globales du classifieur si on lui présentait de nouvelles images (autre que la base de test).
Pour effectuer la validation croisée, on sépare donc la base d'apprentissage en plusieurs parties. Une de ces parties servira au test et le reste à l'apprentissage.

Importer la fonction :
from sklearn.model_selection import cross_val_score
Cette fonction renvoie le score de chaque test effectué :
score = cross_val_score (classifieur, database, data_labels, cv=nombre_de_test)