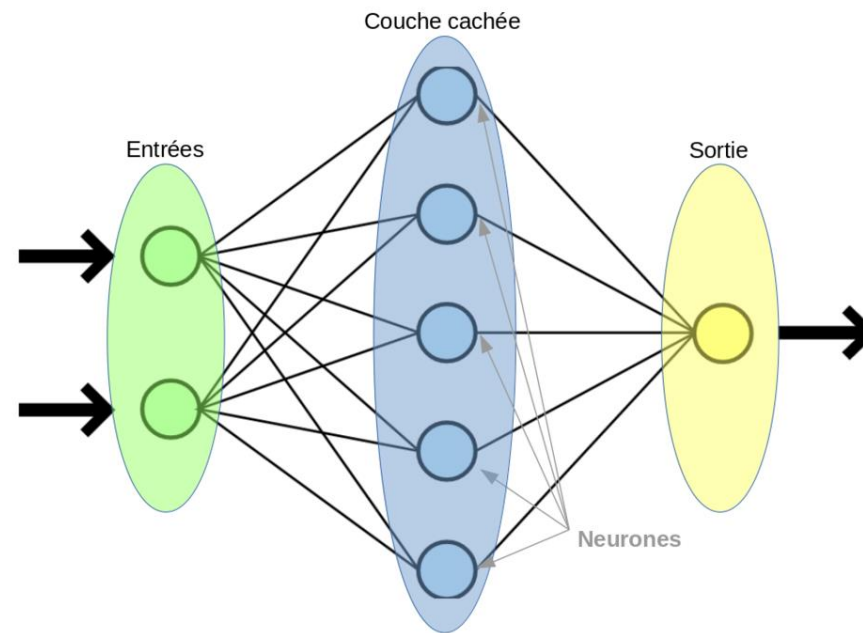


Projet personnel – Système de discrimination des images WANG

Réseaux de neurones

GOLEBIEWSKI ADRIEN



Plus d'informations sur le Repository Github associé : <https://github.com/adriengoleb/Discrimination-Images-Wang>









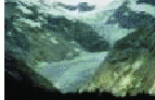

Classification d'une image

Objectif : réaliser **un système de discrimination d'images réelles**. La base de données d'images à notre disposition est une base de données classique de test (base Wang) et est composée de 10 types d'images : Jungle, Plage, Monuments, Bus, Dinosaures, Eléphants, Fleurs, Chevaux, Montagne et Plats.

- L'utilisateur proposera au système une nouvelle image « inconnue »
- le système affichera alors la classe d'appartenance de l'image.

Développement de deux approches :

- l'une reposant sur **des descripteurs caractérisant l'image**
- l'autre suivant **une stratégie « Deep »** donc à base de descripteurs CNN.

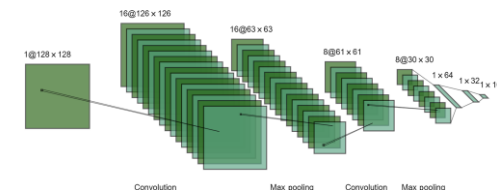
				
Africans	Beaches	Monuments	Buses	Dinosaurs
				
Elephants	Flowers	Horses	Mountains	Foods

Approche Full-connected

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
land	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
beach	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
monument	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
bus	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
dinosaur	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
elephant	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
flower	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
horse	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
mountain	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
food	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
house	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
bird	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
car	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
tree	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
water	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cloud	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
sky	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
grass	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
road	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
city	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

VS

Approche Deep



Classification d'une image – Chargement des descripteurs

Ce fichier propose d'utiliser 5 ensembles de mesures :

JCD,

PHOG,

CEDD,

FCTH,

Fuzzy Color Histogram

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
0.jpg	1.0	4.0	3.0	1.0	2.0	2.0	4.0	3.0	6.0	2.0	1.0	3.0	0.0	0.0	1.0	1.0	
1.jpg	1.0	5.0	5.0	2.0	1.0	4.0	5.0	4.0	6.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
10.jpg	5.0	5.0	2.0	1.0	1.0	3.0	5.0	5.0	5.0	2.0	1.0	0.0	1.0	0.0	1.0	0.0	
100.jpg	1.0	4.0	2.0	2.0	1.0	3.0	2.0	2.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
101.jpg	6.0	6.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	
102.jpg	2.0	6.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	1.0	1.0	0.0	1.0	6.0	
103.jpg	2.0	6.0	1.0	0.0	0.0	0.0	7.0	1.0	4.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
104.jpg	3.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	
105.jpg	0.0	1.0	5.0	1.0	3.0	3.0	5.0	5.0	5.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
106.jpg	5.0	5.0	1.0	0.0	0.0	0.0	6.0	0.0	1.0	3.0	0.0	1.0	0.0	0.0	0.0	0.0	
107.jpg	1.0	4.0	2.0	0.0	0.0	1.0	1.0	0.0	1.0	2.0	0.0	1.0	0.0	0.0	1.0	4.0	
108.jpg	1.0	5.0	2.0	1.0	1.0	2.0	4.0	2.0	4.0	6.0	1.0	5.0	3.0	0.0	3.0	0.0	
109.jpg	1.0	5.0	5.0	0.0	0.0	1.0	3.0	0.0	2.0	2.0	0.0	1.0	0.0	0.0	0.0	1.0	
11.jpg	3.0	3.0	3.0	1.0	1.0	1.0	3.0	2.0	5.0	3.0	3.0	5.0	1.0	0.0	1.0	0.0	
110.jpg	0.0	3.0	3.0	0.0	0.0	2.0	3.0	1.0	5.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
111.jpg	2.0	2.0	4.0	0.0	0.0	0.0	1.0	1.0	2.0	0.0	0.0	3.0	0.0	0.0	2.0	2.0	
112.jpg	2.0	4.0	3.0	2.0	1.0	5.0	3.0	1.0	4.0	2.0	0.0	0.0	0.0	0.0	0.0	2.0	
113.jpg	1.0	4.0	2.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
114.jpg	3.0	3.0	5.0	0.0	0.0	2.0	0.0	0.0	2.0	1.0	0.0	2.0	0.0	0.0	3.0	2.0	
115.jpg	3.0	6.0	4.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	5.0	5.0	
116.jpg	1.0	3.0	2.0	1.0	1.0	3.0	3.0	2.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
117.jpg	5.0	6.0	1.0	0.0	0.0	0.0	2.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	6.0	
118.jpg	4.0	6.0	2.0	0.0	0.0	1.0	2.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
119.jpg	4.0	5.0	3.0	0.0	0.0	0.0	5.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	2.0	0.0	
12.jpg	3.0	3.0	5.0	1.0	3.0	4.0	3.0	3.0	5.0	1.0	1.0	1.0	1.0	0.0	2.0	0.0	
120.jpg	1.0	7.0	2.0	0.0	0.0	0.0	5.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	3.0	



```
#Calcul des descripteurs

#Création du vecteur resultat
target = np.zeros([1,1000])
target = np.array([0]*100+[1]*100+[2]*100+[3]*100+[4]*100+[5]*100+[6]*100+[7]*100+[8]*100+[9]*100)

#association des classes à chaque dataframe
data[0] = pd.to_numeric(data[0].str.rstrip('.jpg'))
data=data.sort_values(by = 0).assign(classe=target)
```

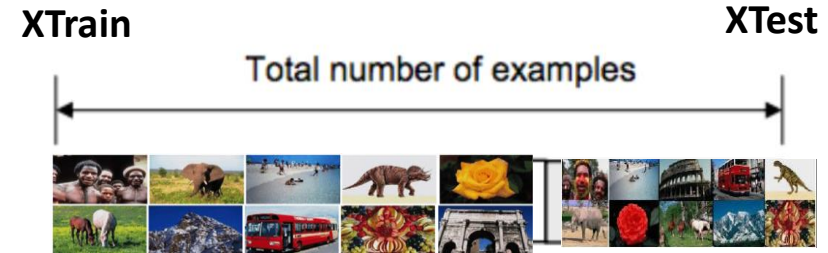
(1) on définit notre tableau cible, la vérité fondamentale de nos données en tant que variable « **target** ».

(1) Mise en ordre des images + suppression des « .jpg » des valeurs de la 1^{ère} colonne numérotant les images. Le label de chacune des images est donc un numéro unique.

Classification d'une image – Approche Full-connected

Approche technique choisie : prédiction pour chaque mesure

Séparation de notre jeu de données : base d'apprentissage Xtrain (80% de notre jeu de données) et un base de test, Xtest (20% de jeux de données).



```
# Pour un modèle avec une entrée et 10 classes (classification en catégories):  
model = Sequential()  
model.add(Dense(270, activation='tanh'))  
model.add(Dense(90, activation='relu'))  
model.add(Dense(60, activation='relu'))  
model.add(Dense(30, activation='relu'))  
model.add(Dense(10, activation='softmax'))
```

```
# fit the keras model on the dataset  
model.fit(XTrain, yTrain, epochs=150, batch_size=100, validation_split=0.1, verbose=0)
```

Mesure PHOG

```
[[10 0 0 0 1 2 0 5 2 0]  
[ 0 13 1 0 0 2 0 2 1 1]  
[ 1 3 10 1 1 0 0 1 2]  
[ 0 0 3 17 0 0 0 0 0]  
[ 0 3 0 0 16 0 0 0 1]  
[ 0 2 1 0 0 11 0 4 1]  
[ 1 0 0 0 0 0 16 0 2]  
[ 0 1 2 0 0 0 0 16 0]  
[ 3 1 0 0 0 1 2 2 9]  
[ 2 1 0 0 0 0 2 2 4 9]]  
0.635
```

Mesure JCD

```
[[15 0 0 0 0 4 0 0 0 1]  
[ 1 17 0 0 0 0 0 0 2]  
[ 0 1 16 1 0 0 0 0 2]  
[ 1 0 0 19 0 0 0 0 0]  
[ 0 0 0 0 20 0 0 0 0]  
[ 0 0 0 0 0 19 0 1 0]  
[ 0 0 0 0 0 0 20 0 0]  
[ 1 0 0 0 0 1 0 18 0]  
[ 0 2 3 0 0 1 0 0 13]  
[ 4 1 0 0 0 0 0 0 15]]  
0.86
```

Mesure CEDD

```
[[16 0 0 0 1 1 0 0 0 2]  
[ 1 18 0 0 0 0 0 0 1]  
[ 1 1 15 0 0 0 0 0 3]  
[ 0 0 0 20 0 0 0 0 0]  
[ 0 0 0 0 20 0 0 0 0]  
[ 1 0 2 0 0 16 0 1 0]  
[ 0 0 0 0 0 0 20 0 0]  
[ 2 0 0 0 0 0 0 18 0]  
[ 0 4 1 0 0 1 1 0 13]  
[ 4 0 0 0 0 1 0 0 15]]  
0.855
```

Mesure FCTH

```
[[13 2 0 0 0 2 0 0 0 3]  
[ 1 15 0 0 0 1 0 0 3]  
[ 0 2 16 1 0 0 0 1 0]  
[ 1 1 0 18 0 0 0 0 0]  
[ 0 0 0 0 20 0 0 0 0]  
[ 0 1 1 0 0 17 0 1 0]  
[ 0 0 0 0 0 0 20 0 0]  
[ 1 0 0 0 0 1 0 18 0]  
[ 0 2 3 0 0 1 0 1 12]  
[ 3 1 0 0 0 0 0 0 16]]  
0.825
```

Mesure FuzzyColor

```
[[10 0 0 0 0 7 1 0 0 2]  
[ 0 6 0 7 0 7 0 0 0]  
[ 5 2 0 9 0 4 0 0 0]  
[ 7 1 0 11 0 0 0 0 1]  
[ 0 0 0 0 19 1 0 0 0]  
[ 0 0 0 0 0 16 0 3 0]  
[ 1 0 0 0 0 2 3 0 14]  
[ 0 0 0 0 0 1 1 18 0]  
[ 1 5 0 11 0 2 0 1 0]  
[ 3 1 0 0 0 1 0 0 15]]  
0.485
```

VS

Concaténation

```
[[ 6 14 0 0 0 0 0 0 0]  
[ 1 19 0 0 0 0 0 0 0]  
[ 0 20 0 0 0 0 0 0 0]  
[ 0 9 0 10 1 0 0 0 0]  
[ 0 1 0 3 15 1 0 0 0]  
[ 0 1 0 0 7 0 6 1 5]  
[ 0 2 0 0 0 0 17 1 0]  
[ 0 3 0 0 0 0 6 6 5]  
[ 0 3 0 0 0 0 0 16 1]  
[ 0 4 0 0 0 0 0 15 1]]  
0.45
```

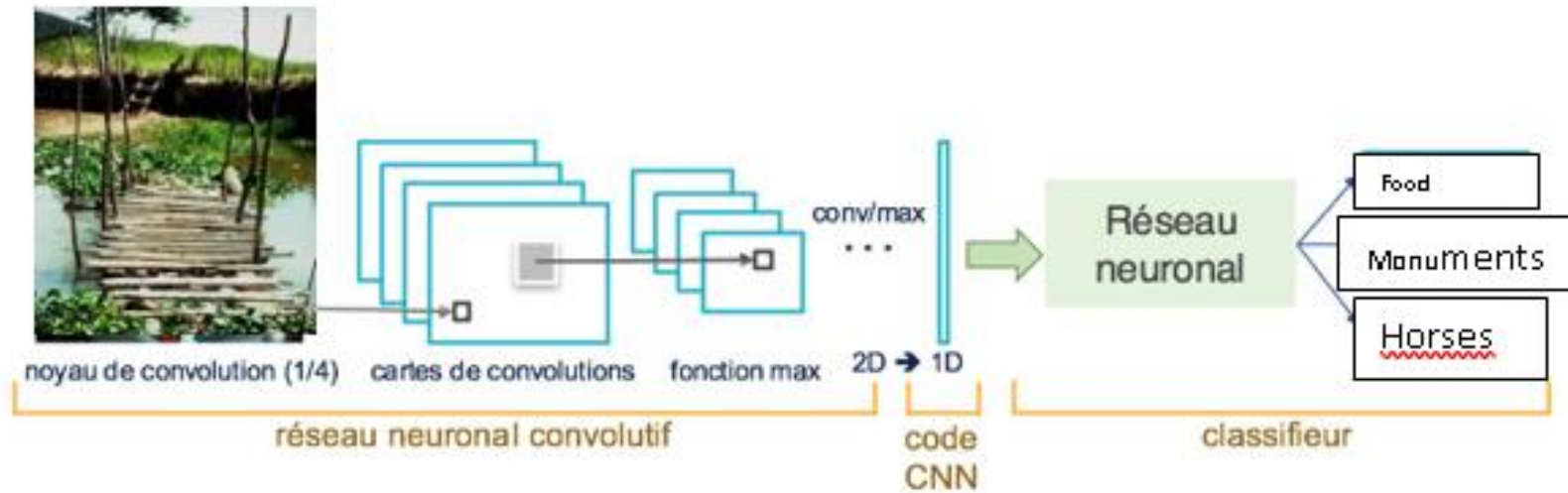
Structure de discrimination de type Perceptron Multi-couches avec Keras/Tensorflow :

- 5 couches (Dernière couche à 10 neurones – 1 pour chaque classe)
- Loss : l'entropie croisée catégorique
- Métrique d'évaluation : accuracy

Classifieur performant avec des scores de précision très correctes. Score de précision le plus élevé avec la mesure JCD (0,86)

Approche technique par mesure plus pertinente que l'approche par concaténation des mesures

Classification d'une image – Approche DeepLearning



- **Les CNN** permettent une extraction de features sans connaissances a priori, et permettent de traiter des données brutes en écartant la phase de feature engineering. Les CNN réalisent eux-mêmes tout le travail fastidieux d'extraction et description de *features*
- **Algorithme CNN sans Data Augmentation :**
 - Data Preparation (images, chemin d'accès, redimensionnement ...)
 - Data Strategy (normalisation des données, split, encodage)
 - Définition du modèle : 2Xconvolution, MaxPooling, un Dropout ,un flatte, un dense et un Dropout → Couches de notre modèle
 - Entraînement du modèle → **Résultat Prédiction** : 0.47 au bout de 3 epochs, >0.60 à parti de 10 epochs

Paramétrage de notre CNN

```
batch_size = 1
num_classes = 10
epochs = 5
img_rows, img_cols = 256, 256
input_shape = (img_rows, img_cols, 3)
e = 2
```

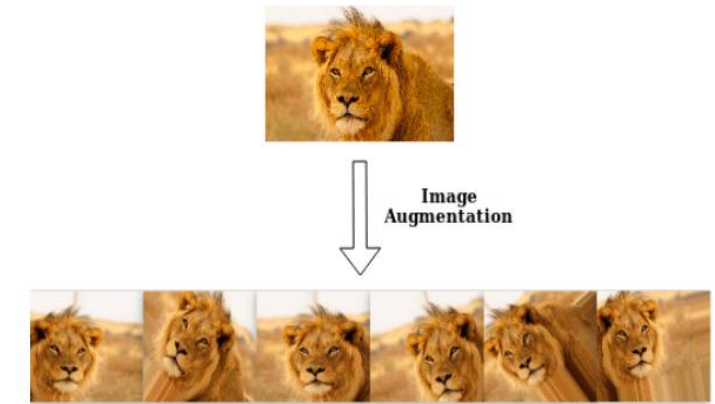
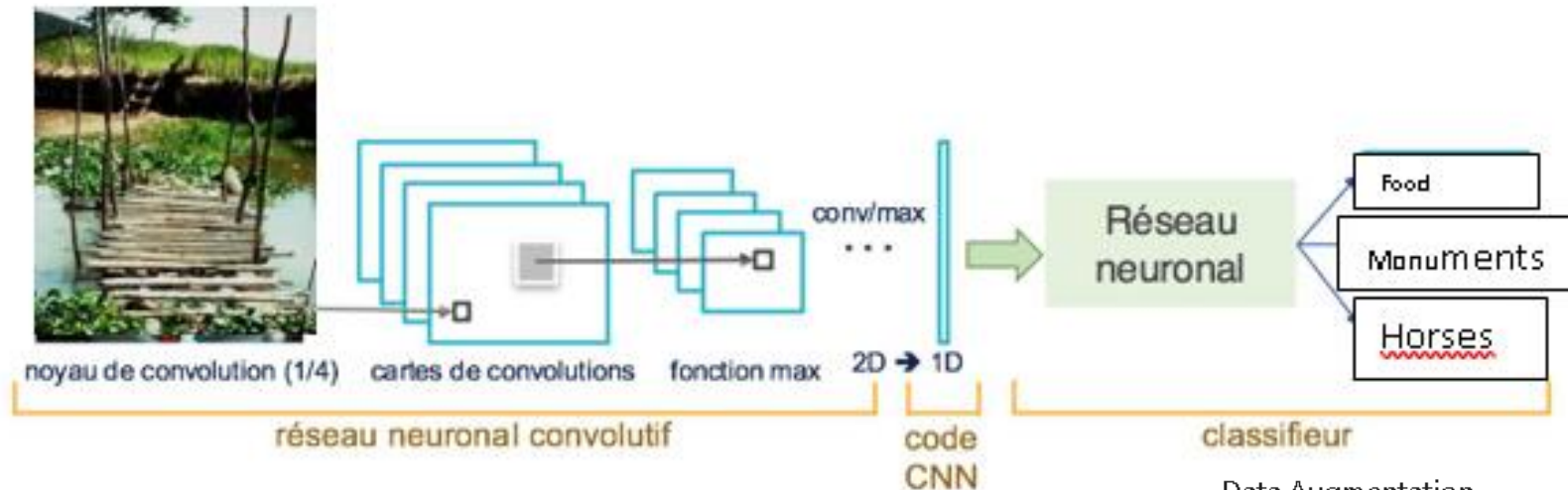
Définition de notre CNN

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape, strides=e))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
```

Apprentissage et Test

```
#Apprentissage du modèle
model.fit(train_images, train_labels,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(test_images, test_labels))
```


Classification d'une image – Approche DeepLearning avec DA



- Mise en place d'un CNN avec Data Augmentation (surplus de données manipulé en amont)

- **Algorithme CNN avec Data Augmentation :**

- Data Preparation / Strategy : même processus
- **Utilisation de « ImageDataGenerator » :** Fonction Keras de génération d'images par Data Augmentation.
- Définition du modèle : 2Xconvolution, MaxPooling, un Dropout ,un flatte, un dense et un Dropout → Couches de notre modèle
- Méthode **fit_generator()** pour ajuster notre modèle sur les données générées par lots par un générateur Python. **Résultats Prédiction : accuracy > 0.85**

Data Augmentation

```
#Augmentation du nombre d'images par la normalisation, la rotation, les décalages,
#Les retournements, le changement de luminosité etc ...

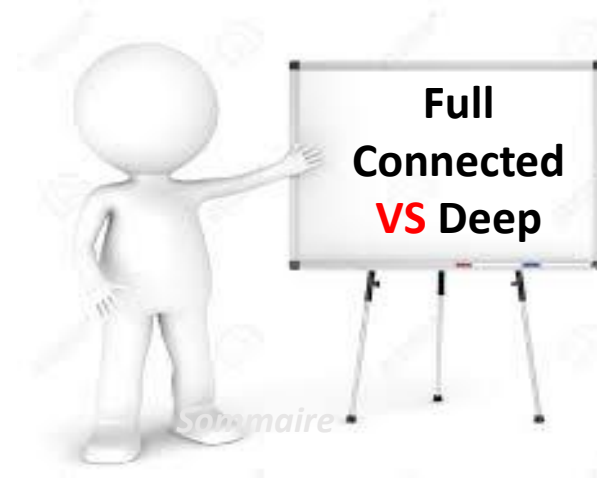
datagen = ImageDataGenerator(
    featurewise_center=False, #fixe la moyenne d'entrée à 0 sur l'ensemble des données
    featurewise_std_normalization=False, #divise les entrées par la moyenne de l'ensemble de données
    samplewise_center=False, #fixe la moyenne de chaque échantillon à 0
    samplewise_std_normalization=False, #divise chaque entrée par sa moyenne
    zca_whitening=False, # application du blanchiment des images
    rotation_range=20, # random rotation des images allant de 0 à 180 degrés
    height_shift_range=0.2, #déplacement aléatoire des images verticalement (fraction de la longueur/hauteur totale)
    width_shift_range=0.2, # déplacement aléatoire des images horizontalement (fraction de la largeur totale)
    horizontal_flip=True, # retournement aléatoire des images à l'horizontal
    vertical_flip=True) # retournement aléatoire des images à la verticale
```

Apprentissage et Test

```
history = model.fit_generator(datagen.flow(train,train_hot, batch_size=5),
                             steps_per_epoch=len(train) / 32,
                             epochs=epochs,validation_data = [test, test_hot])
```

```
25/25 [=====] - 10s 393ms/step - loss: 0.5540 - acc: 0.8125 - val_loss: 0.6794 - val_a
cc: 0.7950
```

Enseignements du projet :



Nous pouvons dès lors affirmer que :

- L'approche « full-connected » se compose d'une série de couches entièrement connectées qui relient chaque neurone d'une couche à chaque neurone de l'autre couche. L'avantage est qu'il n'est pas nécessaire de faire des hypothèses particulières sur l'entrée. Ces types de réseaux ont tendance à avoir des performances plus faibles que les réseaux spécialisés adaptés à la structure d'un espace de problème.
- Les architectures CNN partent de l'hypothèse explicite que les entrées sont des images, ce qui permet d'encoder certaines propriétés de manière automatisée dans l'architecture du modèle. La phase de « feature engineering » est donc volontairement « oubliée »
- Bien que les réseaux « full connected » ne fassent aucune hypothèse sur l'entrée, ils ont tendance à être **moins performants et ne sont pas adaptés à l'extraction de caractéristiques**. De plus, ils ont un nombre plus élevé de poids à former, ce qui entraîne un temps de formation élevé, tandis que les réseaux CNN sont formés pour identifier et extraire les meilleures caractéristiques des images pour le problème à traiter avec relativement moins de paramètres à former.