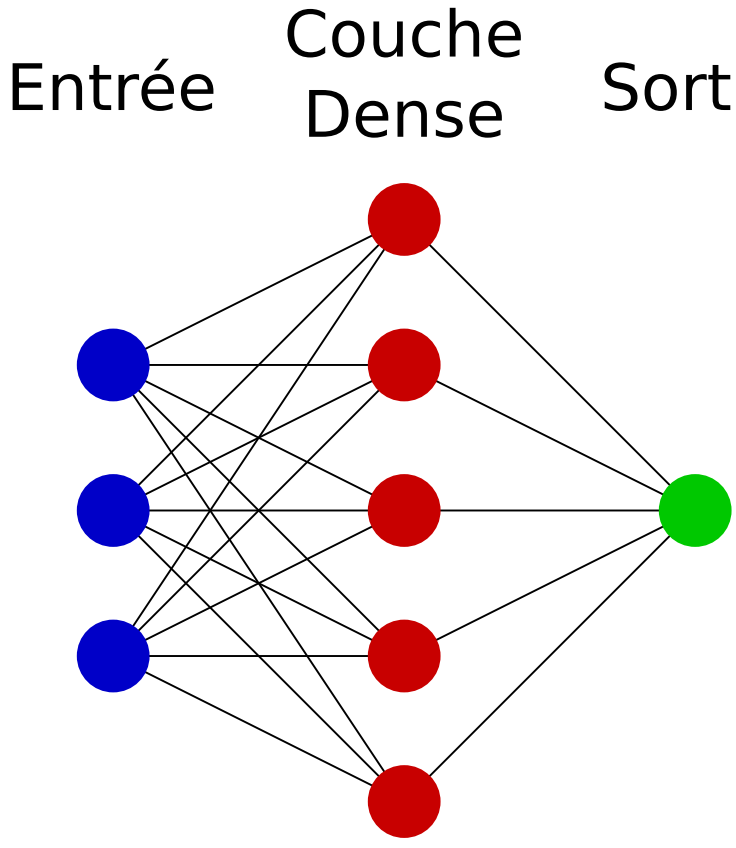
Comment représenter les données avec des outils deep learning pour mieux appréhender nos problèmes ?

Clément Dechesne

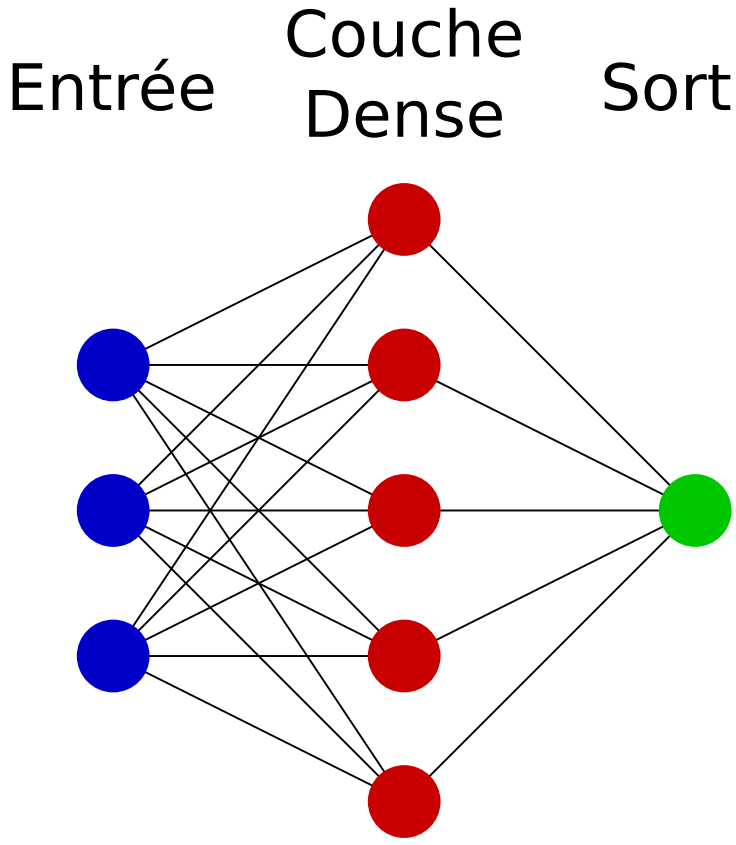
28 Novembre 2018

Les couches denses ou totalement connectées



3 entrées, **5** unités cachées, **15** poids à estimer.

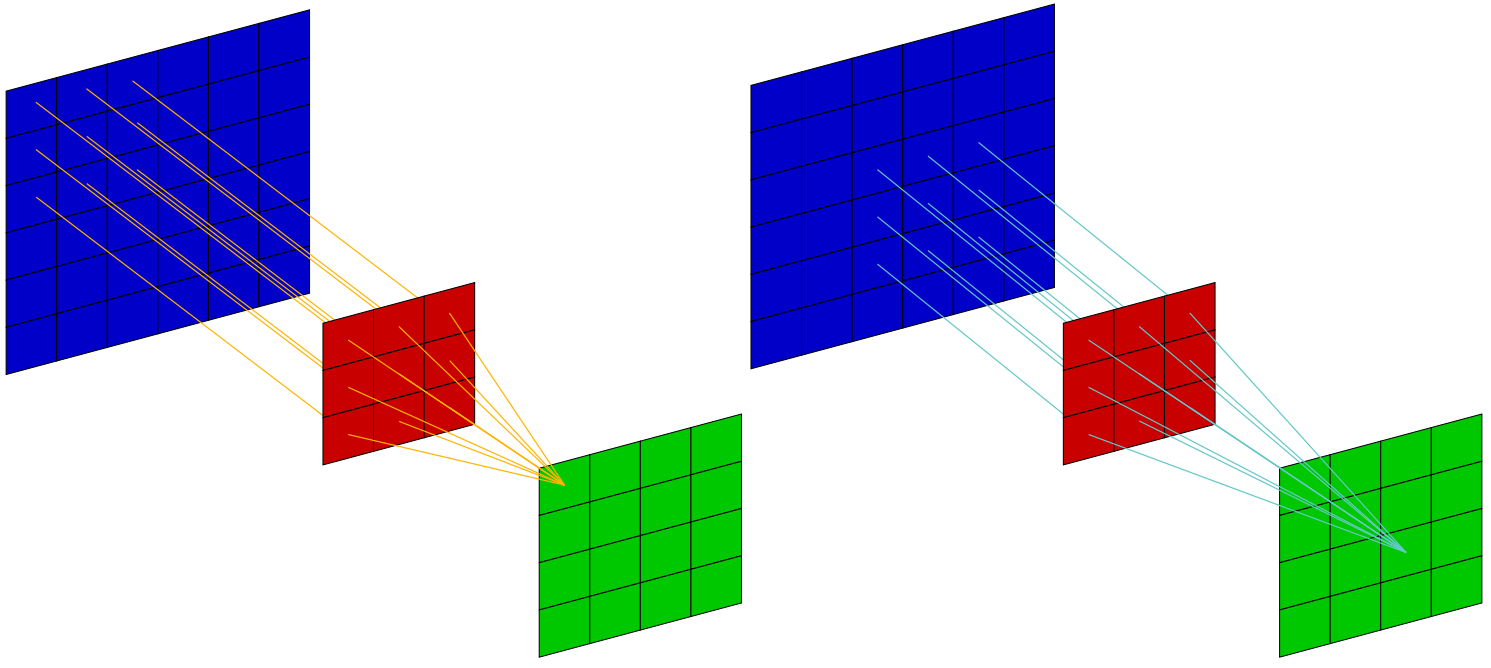
Les couches denses ou totalement connectées



3 entrées, **5** unités cachées, **15** poids à estimer.

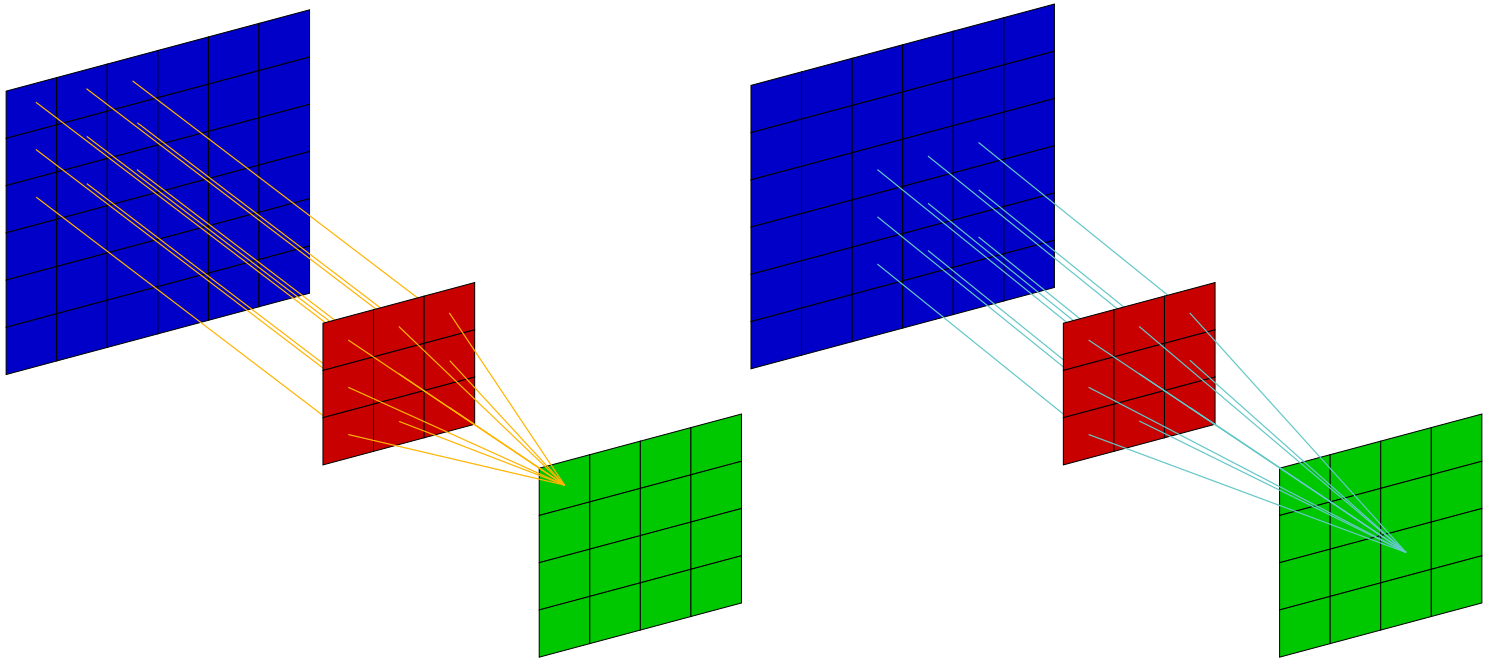
De façon générale, **n** entrées, **m** unités cachées, **n**×**m** poids à estimer.

Convolution



1 Filtre de taille **3**×**3**, **9** poids à estimer.

Convolution



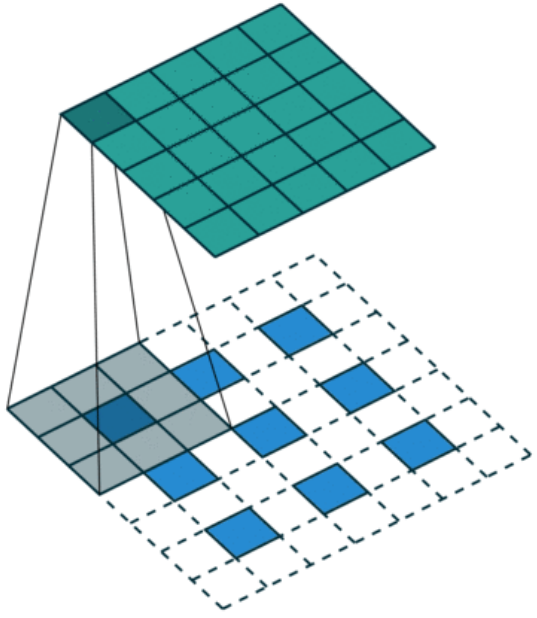
1 Filtre de taille **3**×**3**, **9** poids à estimer.
De façon générale, **n** filtres de taille **m**×**m**,
n×**m**×**m** poids à estimer.

Pooling

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

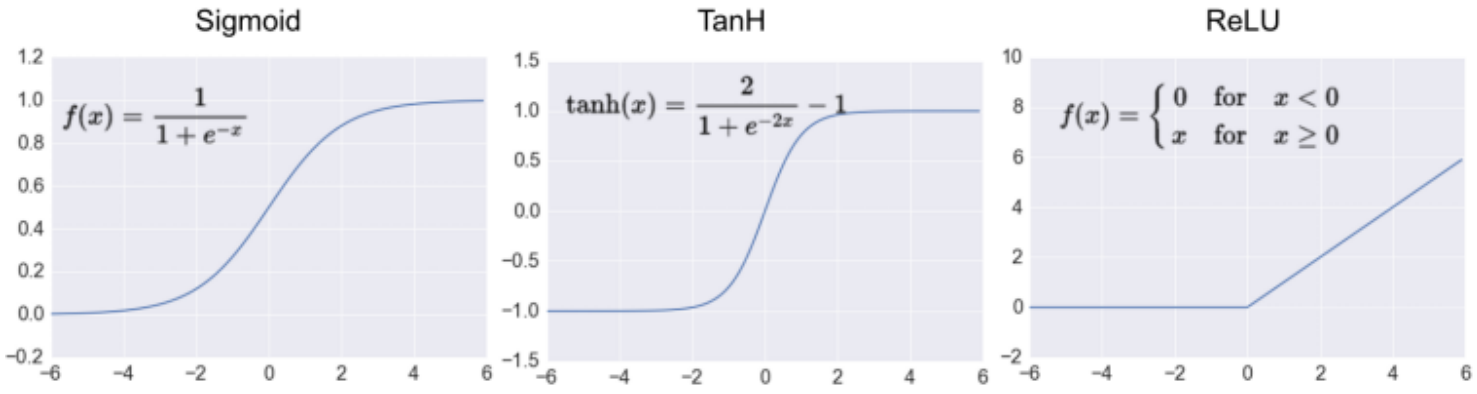
2×2 Max-Pool →

20	30
112	37



Activation

Après une couche dense ou de convolution, il est possible d'appliquer une fonction (l'activation) aux sorties afin d'ajouter de la non-linéarité au problème.



A quoi ça sert ?

Autoencoder - Généralités

A quoi ça sert ?

Réduire la dimension, en gardant l'information pertinente.

Autoencoder - Généralités

A quoi ça sert ?

Réduire la dimension, en gardant l'information pertinente.

Quelle type de donnée ?

Autoencoder - Généralités

A quoi ça sert ?

Réduire la dimension, en gardant l'information pertinente.

Quelle type de donnée ?

Tous.

Autoencoder - Généralités

A quoi ça sert ?

Réduire la dimension, en gardant l'information pertinente.

Quelle type de donnée ?

Tous.

Les paramètres ?

Autoencoder - Généralités

A quoi ça sert ?

Réduire la dimension, en gardant l'information pertinente.

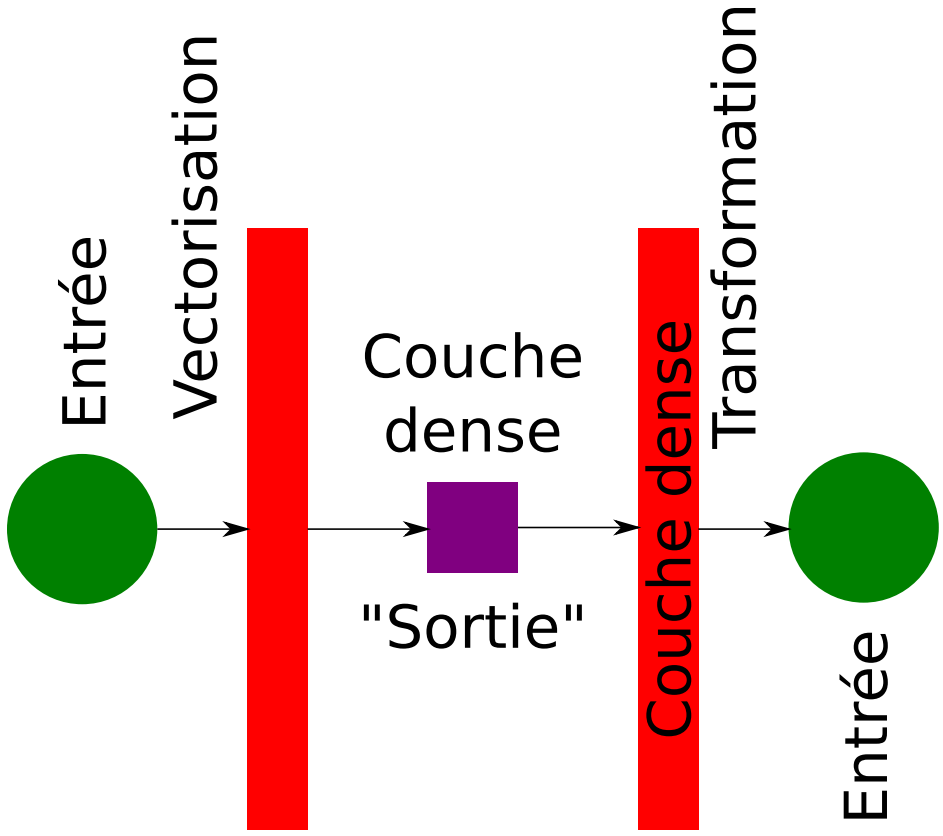
Quelle type de donnée ?

Tous.

Les paramètres ?

Un seul, la dimension voulue.

Autoencoder - Architecture



Chargement des librairies

In [1]:

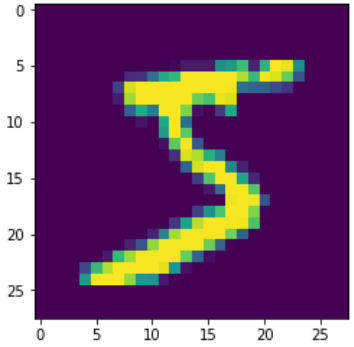
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import keras
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, load_model, Input, Model
from keras.layers import Dense, Dropout, UpSampling2D, Activation, Flatten, Lambda, Reshape
from keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose, AveragePooling2D
from keras.layers import Concatenate, Add, Multiply, Reshape, Dot
from keras.layers import LSTM

from keras.callbacks import TensorBoard

from keras.datasets import mnist

import numpy as np
import os, shutil, sys
import imageio
import random
import matplotlib.pyplot as plt
from matplotlib import cm
import statistics
import datetime
import time
import spectral.io.envi as envi
import imageio

from sklearn.svm import SVC
```

```
Using TensorFlow backend.
```

Base de donnée MNIST ¶

In [2]:
```
(x_train_t, y_train), (x_test_t, y_test) = mnist.load_data()

N1=x_train_t.shape[0]
N2=x_test_t.shape[0]
I=x_train_t.shape[1]
J=x_train_t.shape[2]

x_train=np.zeros((N1,I,J,1))
x_test=np.zeros((N2,I,J,1))

x_train[:,:,:,0]=x_train_t/255.0
x_test[:,:,:,0]=x_test_t/255.0

num_classes=np.max(y_train)+1

y_train_class=keras.utils.to_categorical(y_train, num_classes=num_classes)
y_test_class=keras.utils.to_categorical(y_test, num_classes=num_classes)
```

In [3]:
```
id_sample=0

plt.imshow(x_train[id_sample,:,:,0])
plt.show()

print(y_train[id_sample])
```




```
5
```

Modèles

In [5]:
```
I=x_train.shape[1]
J=x_train.shape[2]
L=x_train.shape[3]

encoding_dim = 20

I1=Input(shape=(I,J,L),name="Input1")
F=Flatten(name="Flatten")(I1)
encoder=Dense(encoding_dim)(F)
A1=Activation("relu")(encoder)

A2=Dense(num_classes,activation="softmax")(A1)

I2=Dense(I*J*L,activation="sigmoid")(A1)
decoder=Reshape((I, J, L))(I2)

model_feature=Sequential()
model_feature=Model(inputs=[I1], outputs=[A1])

model_classif=Sequential()
model_classif=Model(inputs=[I1], outputs=[A2])

model_autoencoder=Sequential()
model_autoencoder=Model(inputs=[I1], outputs=[decoder])

# initiate RMSprop optimizer
opt=keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)

losses=["categorical_crossentropy"]
employed_metrics=["mae","mse","accuracy"]
employed_weights=[1.0]
model_classif.compile(optimizer=opt, loss=losses, metrics=employed_metrics, loss_weights=employed_weights)

losses=["mse"]
employed_metrics=["mae","accuracy"]
employed_weights=[1.0]
model_feature.compile(optimizer=opt, loss=losses, metrics=employed_metrics, loss_weights=employed_weights)

losses=["mse"]
employed_metrics=["mae","accuracy"]
employed_weights=[1.0]
model_autoencoder.compile(optimizer=opt, loss=losses, metrics=employed_metrics, loss_weights=employed_weights)
```

```
Autoencoder:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input1 (InputLayer)          (None, 28, 28, 1)         0
_________________________________________________________________
Flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                15700
_________________________________________________________________
activation_1 (Activation)    (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 784)               16464
_________________________________________________________________
reshape_1 (Reshape)          (None, 28, 28, 1)         0
=================================================================
Total params: 32,164
Trainable params: 32,164
Non-trainable params: 0
_________________________________________________________________
Total params: 32164
Trainable params: 32164
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
Classification:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input1 (InputLayer)          (None, 28, 28, 1)         0
_________________________________________________________________
Flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                15700
_________________________________________________________________
activation_1 (Activation)    (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 15,910
Trainable params: 15,910
Non-trainable params: 0
_________________________________________________________________
Total params: 15910
Trainable params: 15910
Non-trainable params: 0
```

Entrainement des modèles

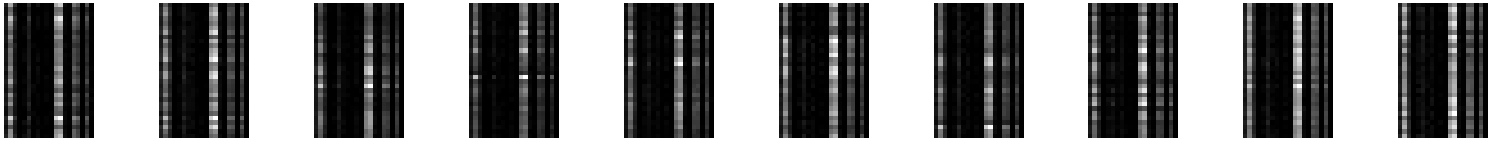
In [6]:

```
batch_size=100
epochs=100

tensorboard_autoencoder = TensorBoard(log_dir='./logs_autoencoder', histogram_freq=2, batch_size=batch_size, write_graph=T
                          write_images=True)

tensorboard_classif = TensorBoard(log_dir='./logs_classif', histogram_freq=2, batch_size=batch_size, write_graph=True, wri
                          write_images=True)

#"""
model_autoencoder.fit([x_train],[x_train],batch_size=batch_size,epochs=epochs, verbose=1, validation_split=0.2,callbacks=[

model_feature.save("model_feature_autoencoder.h5")
model_classif.save("model_classif_autoencoder.h5")
model_autoencoder.save("model_autoencoder_autoencoder.h5")
#"""

#"""
model_classif.fit([x_train],[y_train_class],batch_size=batch_size,epochs=epochs, verbose=1, validation_split=0.2,callbacks=

model_feature.save("model_feature_classif.h5")
model_classif.save("model_classif_classif.h5")
model_autoencoder.save("model_autoencoder_classif.h5")
#"""
```

Résultats - Features

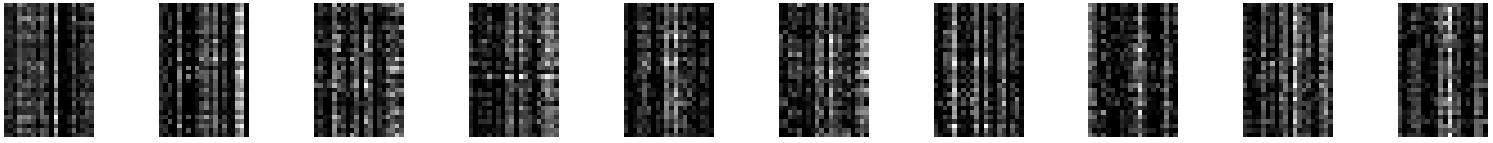
Dimension de l'autoencoder : 20

Autoencoder



0 1 2 3 4 5 6 7 8 9

Autoencoder - Classification



0 1 2 3 4 5 6 7 8 9

Résultats - Classification SVM

Autoencoder : Précision finale avec un SVM à noyau gaussien (C=1000, gamma=0.02) : **83.54%**

Autoencoder classification : Précision finale avec un SVM à noyau gaussien (C=1, gamma=0.02) : **96.62%**

A quoi ça sert ?

Généralités

A quoi ça sert ?

Extraire de l'information (des attributs).

Généralités

A quoi ça sert ?

Extraire de l'information (des attributs).

Quelle type de donnée ?

Généralités

A quoi ça sert ?

Extraire de l'information (des attributs).

Quelle type de donnée ?

Images.

Généralités

A quoi ça sert ?

Extraire de l'information (des attributs).

Quelle type de donnée ?

Images.

Les paramètres ?

Généralités

A quoi ça sert ?

Extraire de l'information (des attributs).

Quelle type de donnée ?

Images.

Les paramètres ?

Le nombre d'attributs voulus par pixel.

Architecture

Ce type de réseau est composé de deux parties :

- Une partie qui permet de réduire "la taille" de l'image, en enchaînant couches convolutionelles et couches de pooling.
- Une seconde partie qui augmente "la taille" de l'image, afin de retrouver la taille originale, en enchaînant couches convolutionelles et couches de deconvolution.

Modèle ¶

In [3]:

```
NFeat=20
FConv=64
FDConv=128

I1=Input(shape=(None,None,L),name="Input")

Conv01=Conv2D(FConv, (3, 3), padding='same',name="CLayer1",activation="relu")(I1)
Conv02=Conv2D(FConv, (3, 3), padding='same',name="CLayer2",activation="relu")(Conv01)
MaxPool01=MaxPooling2D(pool_size=(2, 2))(Conv02)

Conv03=Conv2D(FConv, (3, 3), padding='same',name="CLayer3",activation="relu")(MaxPool01)
Conv04=Conv2D(FConv, (3, 3), padding='same',name="CLayer4",activation="relu")(Conv03)
MaxPool02=MaxPooling2D(pool_size=(2, 2))(Conv04)

Conv05=Conv2D(FConv, (3, 3), padding='same',name="CLayer5",activation="relu")(MaxPool02)
Conv06=Conv2D(FConv, (3, 3), padding='same',name="CLayer6",activation="relu")(Conv05)

DConv1=Conv2DTranspose(FDConv, (3, 3), strides=(2, 2), padding='same',name="DCLayer1",activation="relu")(Conv06)
Conv07=Conv2D(FConv, (3, 3), padding='same',name="CLayer7",activation="relu")(DConv1)
Conv08=Conv2D(FConv, (3, 3), padding='same',name="CLayer8",activation="relu")(Conv07)

DConv2=Conv2DTranspose(FDConv, (3, 3), strides=(2, 2), padding='same',name="DCLayer2",activation="relu")(Conv08)
Conv09=Conv2D(FConv, (3, 3), padding='same',name="CLayer9",activation="relu")(DConv2)
Conv10=Conv2D(FConv, (3, 3), padding='same',name="CLayer10",activation="relu")(Conv09)

O2=Conv2D(NFeat, (3, 3), padding='same',name="Output2")(Conv10)
O2a=Activation("relu")(O2)

O1=Conv2D(num_classes, (3, 3), padding='same',name="Output1")(O2a)
O1a=Activation("softmax")(O1)

model=Sequential()
model=Model(inputs=[I1], outputs=[O1a])

model_feature=Sequential()
model_feature=Model(inputs=[I1], outputs=[O2])

# initiate RMSprop optimizer
opt=keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)

losses=["categorical_crossentropy"]
employed_metrics=["mae","mse","accuracy"]
employed_weights=[1.0]
model.compile(optimizer=opt, loss=losses, metrics=employed_metrics, loss_weights=employed_weights)

losses=["mse"]
employed_metrics=["mae","accuracy"]
employed_weights=[1.0]
model_feature.compile(optimizer=opt, loss=losses, metrics=employed_metrics, loss_weights=employed_weights)
```

Modèle de classification:

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, None, None, 3)	0
CLayer1 (Conv2D)	(None, None, None, 64)	1792
CLayer2 (Conv2D)	(None, None, None, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, None, None, 64)	0
CLayer3 (Conv2D)	(None, None, None, 64)	36928
CLayer4 (Conv2D)	(None, None, None, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, None, None, 64)	0
CLayer5 (Conv2D)	(None, None, None, 64)	36928
CLayer6 (Conv2D)	(None, None, None, 64)	36928
DCLayer1 (Conv2DTranspose)	(None, None, None, 128)	73856
CLayer7 (Conv2D)	(None, None, None, 64)	73792
CLayer8 (Conv2D)	(None, None, None, 64)	36928
DCLayer2 (Conv2DTranspose)	(None, None, None, 128)	73856
CLayer9 (Conv2D)	(None, None, None, 64)	73792
CLayer10 (Conv2D)	(None, None, None, 64)	36928
Output2 (Conv2D)	(None, None, None, 20)	11540
activation_1 (Activation)	(None, None, None, 20)	0
Output1 (Conv2D)	(None, None, None, 12)	2172
activation_2 (Activation)	(None, None, None, 12)	0

Total params: 569,296
Trainable params: 569,296
Non-trainable params: 0

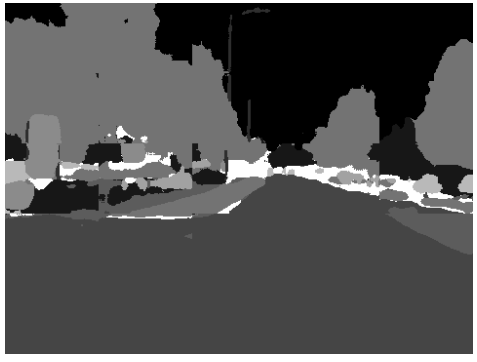
Total params: 569296
Trainable params: 569296
Non-trainable params: 0

Résultats - Classification par le réseau

Image

Vérité terrain

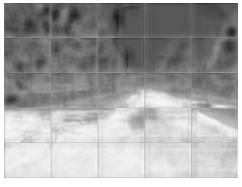
Classification



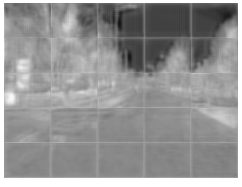
Résultats - Attributs

Image

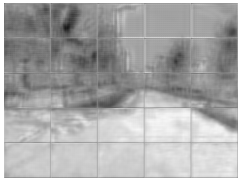
1



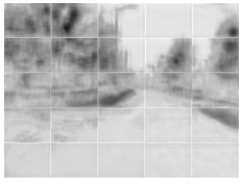
2



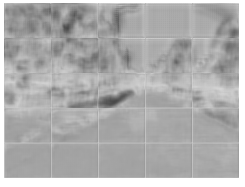
3



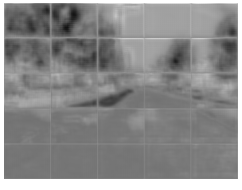
4



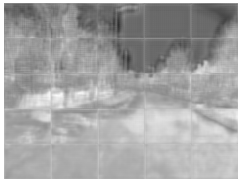
5



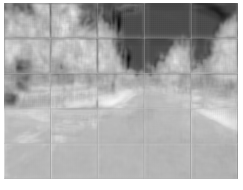
6



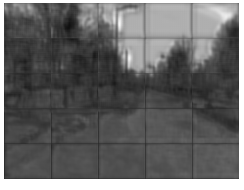
7



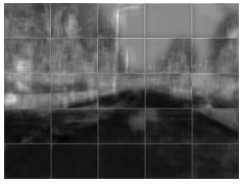
8



9



10



Comment représenter les données avec des outils deep learning pour mieux appréhender nos problèmes ?

Clément Dechesne

28 Novembre 2018