Analyse d'images sportives

TIPE - Jeux et sports

Thomas KUMMEL - 12188

Année 2023-2024

Sommaire

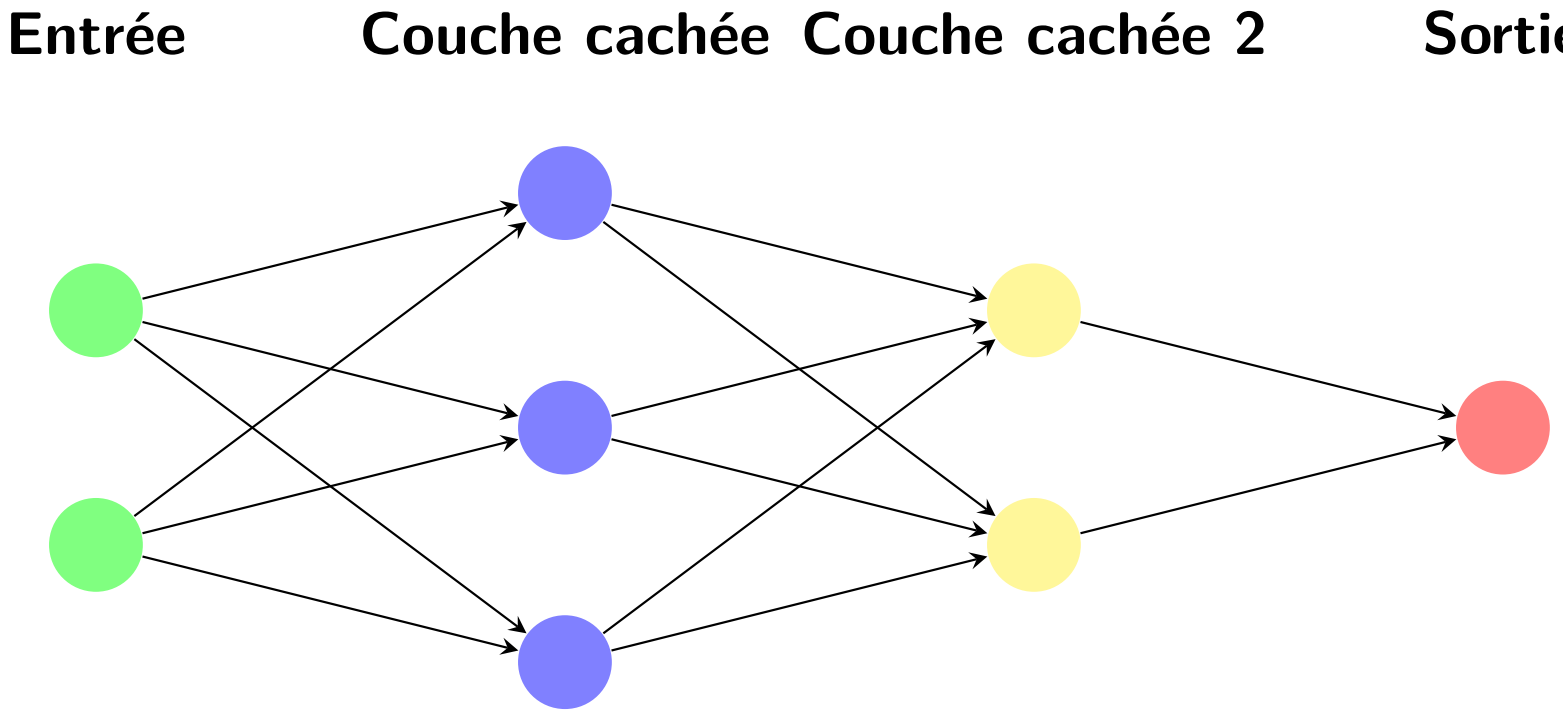
Sommaire

Projet

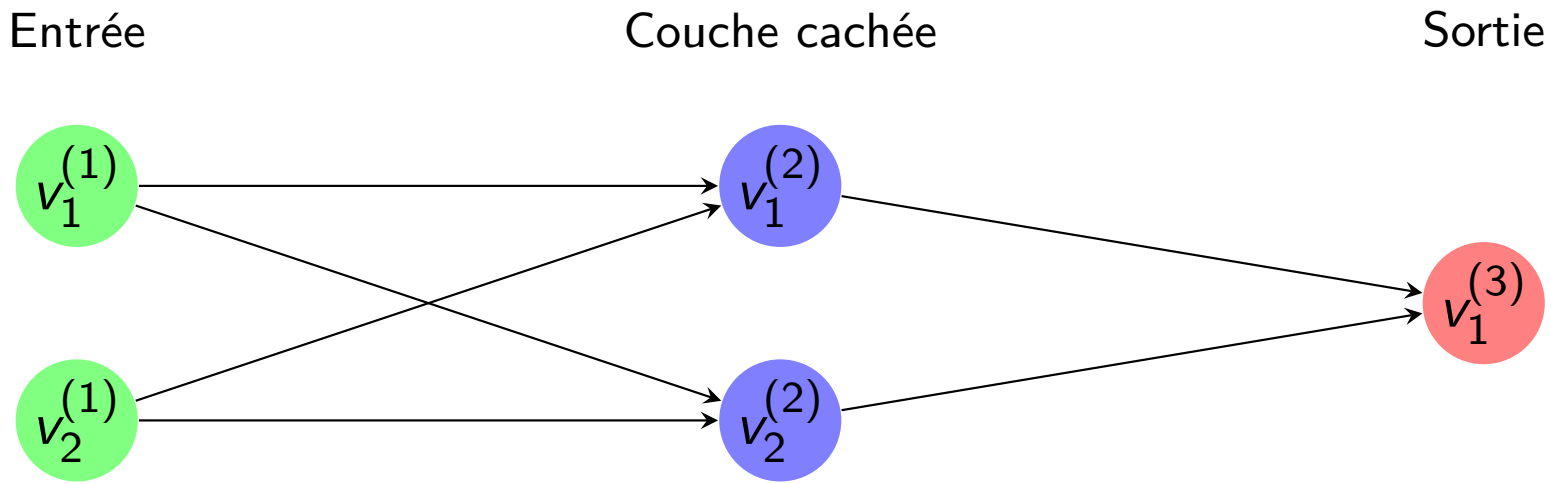
Est-il possible d'utiliser un réseau de neurones pour l'analyse d'images sportives ?

Sommaire

Perceptron multicouches

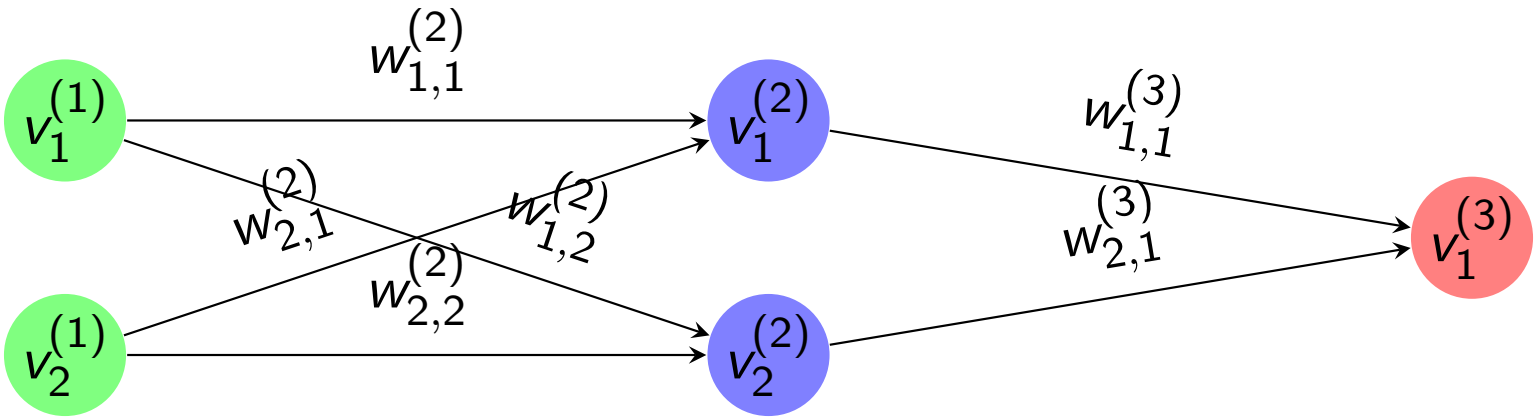


Définitions I



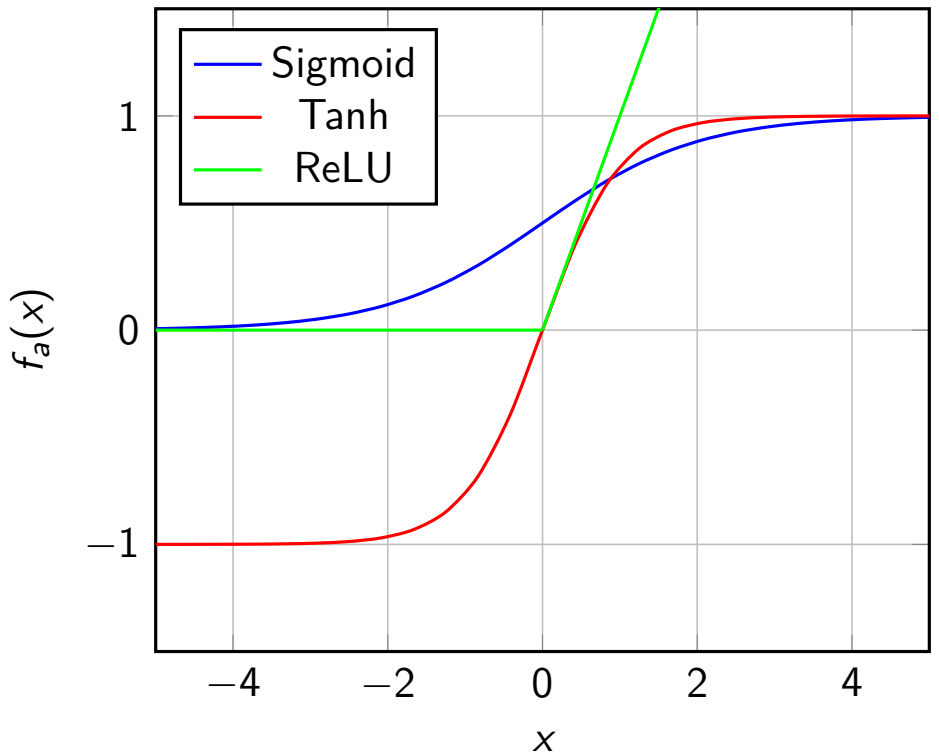
- **Neurone** $v_i^{(k)}$: élément qui prend une valeur en entrée pour en renvoyer une autre ;
- **Couche** k : rangée de neurones de 3 types : entrée, cachée, sortie ;
- **Dimension** c_k : nombre de neurones dans une couche k ;

Définitions II



- **Poids** $w^{(k)}_{i,j}$: facteur qui amplifie ou atténue les connexions ;
- **Biais** $b^{(k)}_i$: décale la fonction d'activation ;

Définitions III



➡ **Fonction d'activation** : non-linéarités dans le réseau.

Définitions IV

- Valeurs :

$$V_k = \begin{pmatrix} v_1^{(k)} \\ \vdots \\ v_{c_k}^{(k)} \end{pmatrix}$$

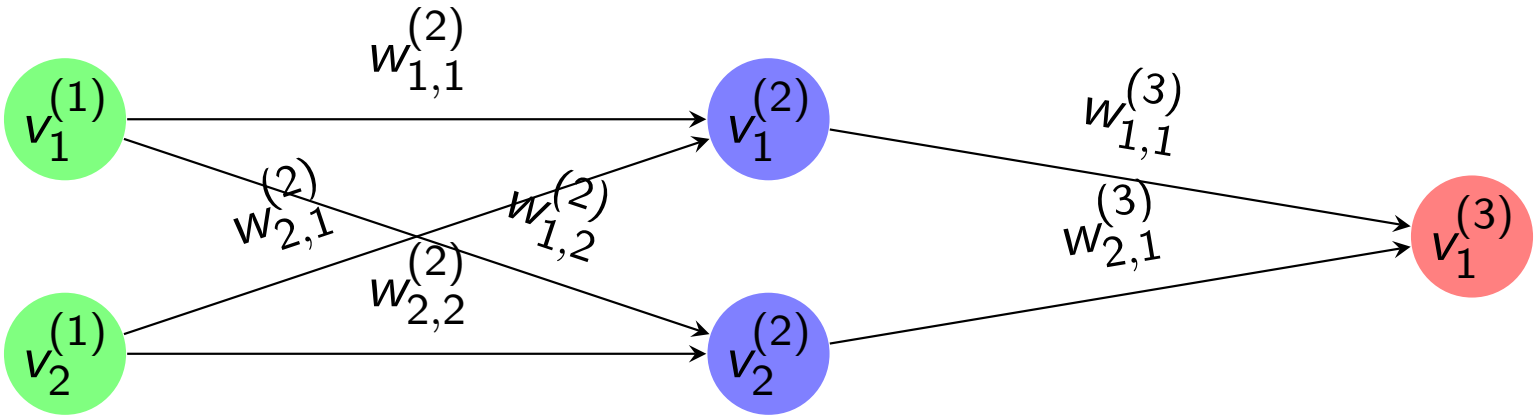
- Biais :

$$B_k = \begin{pmatrix} b_1^{(k)} \\ \vdots \\ b_{c_k}^{(k)} \end{pmatrix}$$

- Poids :

$$W_k = \begin{pmatrix} w_{0,0}^{(k)} & \cdots & w_{c_{k-1},0}^{(k)} \\ \vdots & \vdots & \vdots \\ w_{0,c_k}^{(k)} & \cdots & w_{c_{k-1},c_k}^{(k)} \end{pmatrix}$$

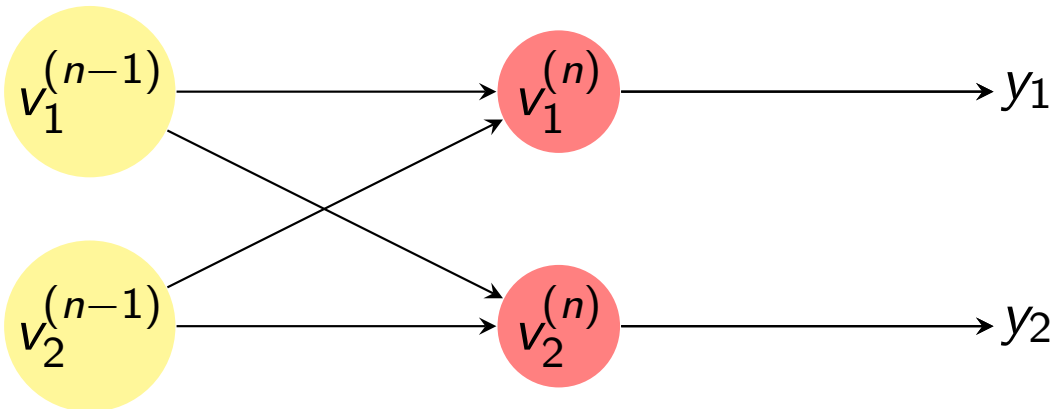
Propagation



$$v_i^{(k)} = f_a\left(\sum_{j=1}^{c_{k-1}} v_j^{(k-1)} \times w_{j,i}^{(k)} + b_i^{(k)}\right) \qquad V_k = f_a\left(W_k \cdot V_{k-1} + B_k\right)$$

Nom	Dimensions	Valeurs	Poids	Biais
Variable	$(c_k)_{1 \leq k \leq n}$	$v_i^{(k)}$	$w_{i,j}^{(k)}$	$b_i^{(k)}$

Erreur d'un réseau



$$C = \sum_{i=1}^{c_n} \frac{1}{2} \left(v_i^{(n)} - y_i \right)^2$$

Nom	Dimensions	Valeurs	Poids	Biais
Variable	$(c_k)_{1 \leq k \leq n}$	$v_i^{(k)}$	$w_{i,j}^{(k)}$	$b_i^{(k)}$

Descente de gradient

- Taux d'apprentissage : α ;
- Correction d'un réseau de neurones

$$w_{i,j}^{(n)} = w_{i,j}^{(n)} - \alpha \cdot \frac{\partial C}{\partial w_{i,j}^{(n)}}$$

$$b_i^{(n)} = b_i^{(n)} - \alpha \frac{\partial C}{\partial b_i^{(n)}}$$

Nom	Dimensions	Valeurs	Poids	Biais
Variable	$(c_k)_{1 \leq k \leq n}$	$v_i^{(k)}$	$w_{i,j}^{(k)}$	$b_i^{(k)}$

Procédure de fonctionnement

1. Initialisation des poids et des biais ;
2. Pour chaque donnée du paquet d'entraînement :
   - 2.1 Propagation de l'information ;
   - 2.2 Calcul du coût de l'entraînement ;
   - 2.3 Algorithme de descente de gradient ;
   - 2.4 Correction des poids et des biais ;
3. Obtention de prédictions correctes.

MNIST

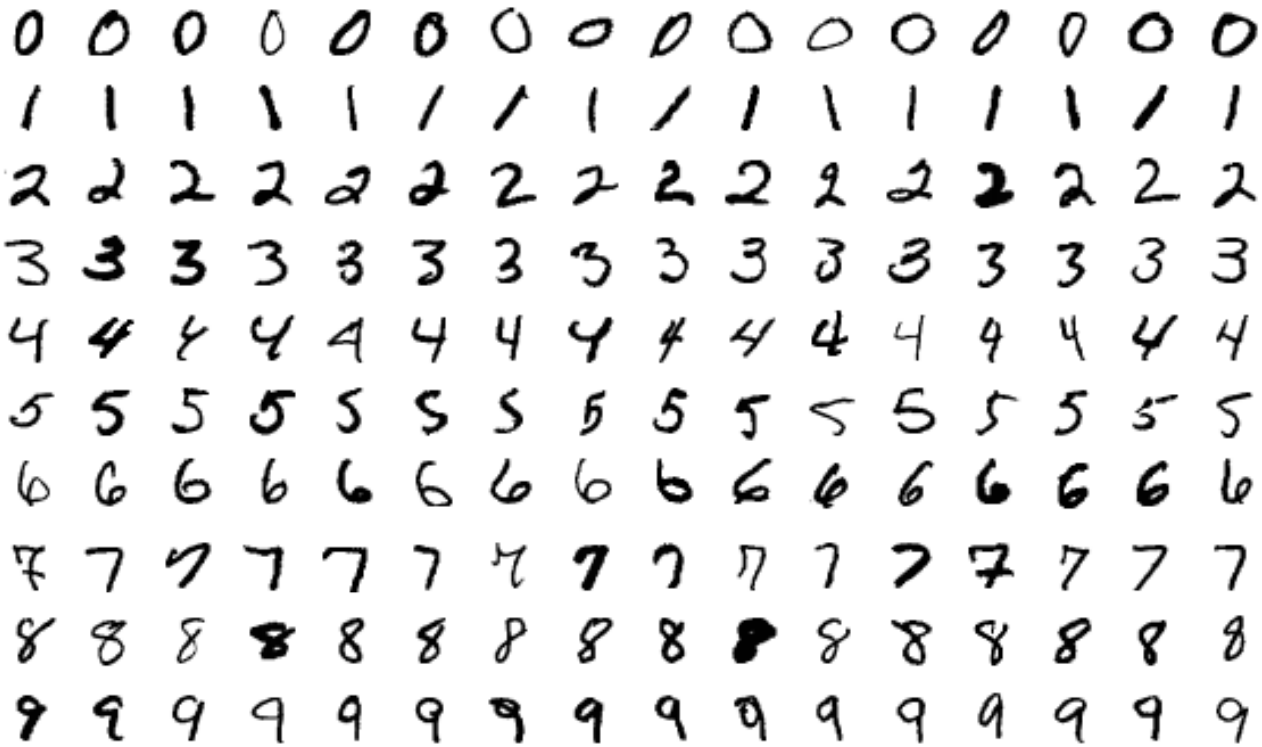
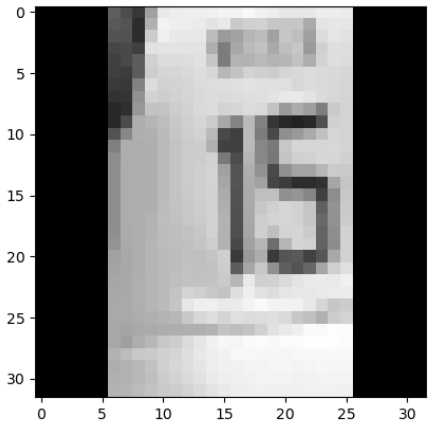
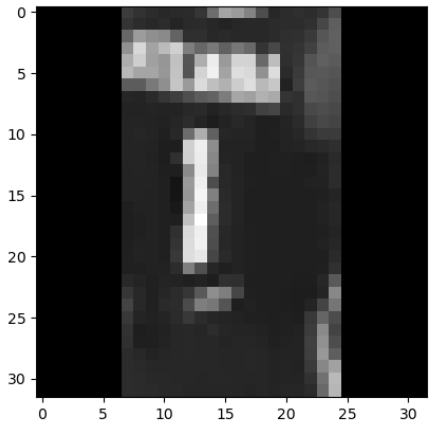


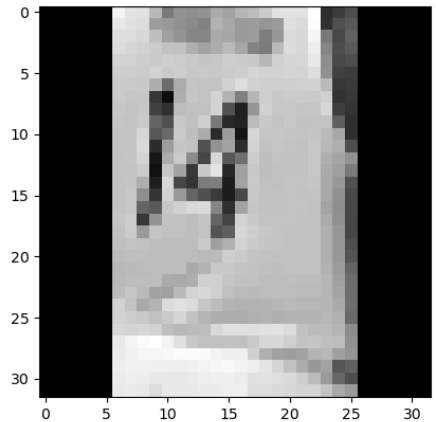
Figure – Exemples de données de MNIST
Source : wikipedia.org

Sommaire

Base de données







Source : Matteo BORGHESI, kaggle.com

Propagation

Filtre F_k

$f_{1,1}$	$f_{1,2}$	$f_{1,3}$
$f_{2,1}$	$f_{2,2}$	$f_{2,3}$
$f_{3,1}$	$f_{3,2}$	$f_{3,3}$

Convolution

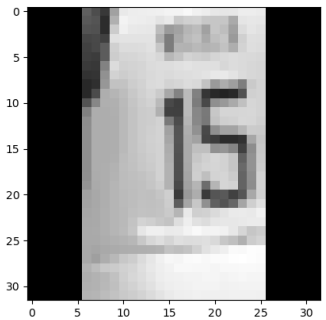
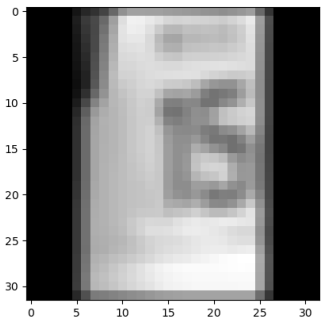
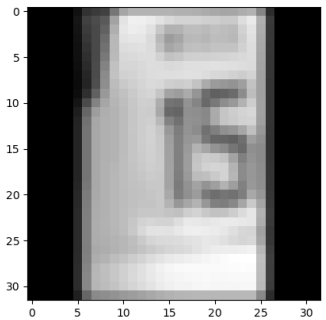
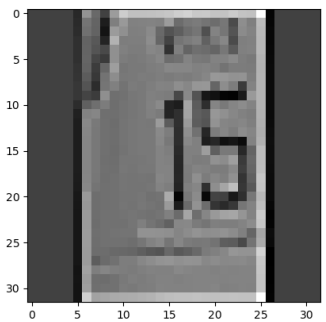
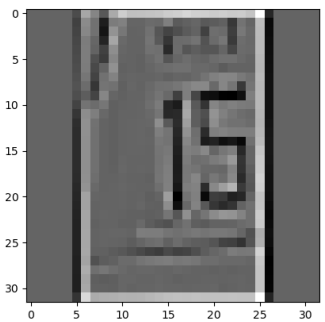
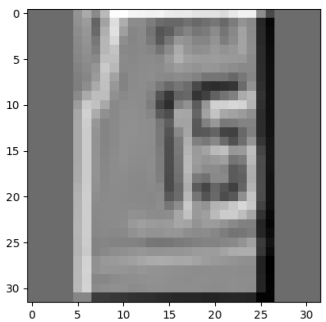
$v_{1,1}^{(k)} \times f_{3,3}$	$v_{1,2}^{(k)} \times f_{3,2}$	$v_{1,3}^{(k)} \times f_{3,1}$	$v_{1,4}^{(k)}$
$v_{2,1}^{(k)} \times f_{2,3}$	$v_{2,2}^{(k)} \times f_{2,2}$	$v_{2,3}^{(k)} \times f_{2,1}$	$v_{2,4}^{(k)}$
$v_{3,1}^{(k)} \times f_{1,3}$	$v_{3,2}^{(k)} \times f_{1,2}$	$v_{3,3}^{(k)} \times f_{1,1}$	$v_{3,4}^{(k)}$
$v_{4,1}^{(k)}$	$v_{4,2}^{(k)}$	$v_{4,3}^{(k)}$	$v_{4,4}^{(k)}$

Résultat V_k

$v_{1,1}^{(k)}$	$v_{1,2}^{(k)}$
$v_{2,1}^{(k)}$	$v_{2,2}^{(k)}$

Nom	Dimensions	Valeurs	Poids	Biais	Filtre
Variable	$(c_k)_{1 \leq k \leq n}$	$v_i^{(k)}$	$w_{i,j}^{(k)}$	$b_i^{(k)}$	$f_{i,j}^{(k)}$

Filtres de convolution

Résultats

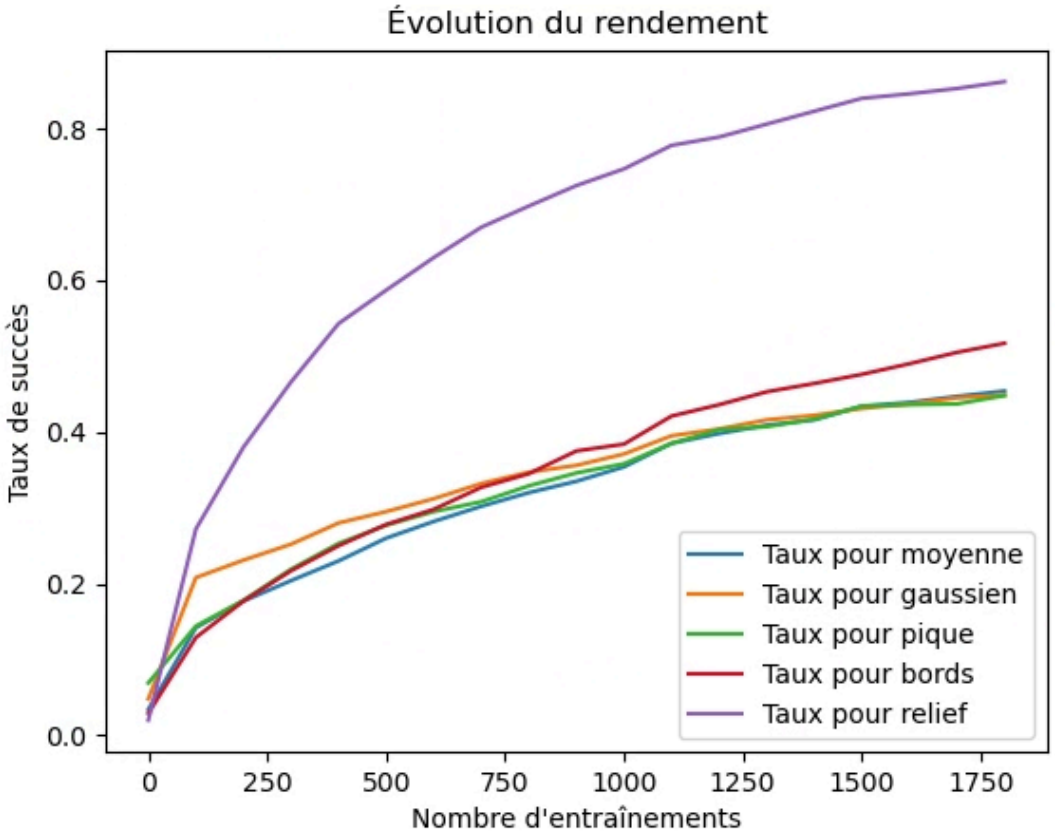


Figure – Résultats du perceptron multicouches

Convolution à 1 niveau

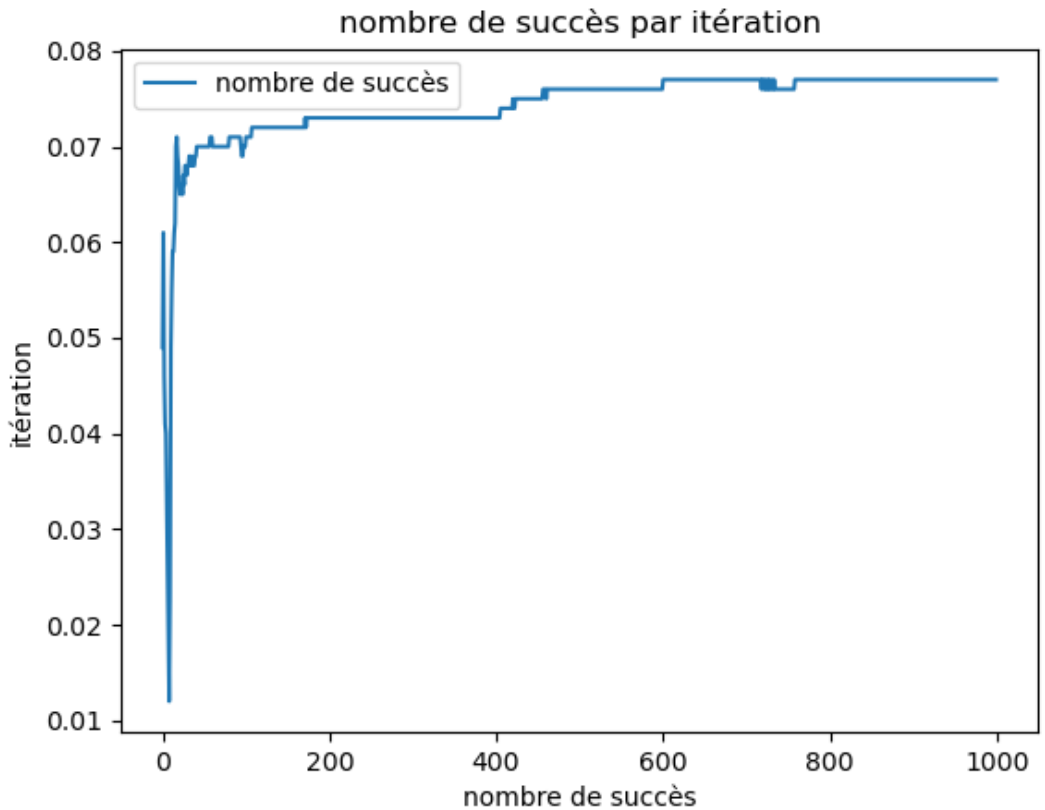


Figure – Résultats

Convolution à plusieurs niveaux

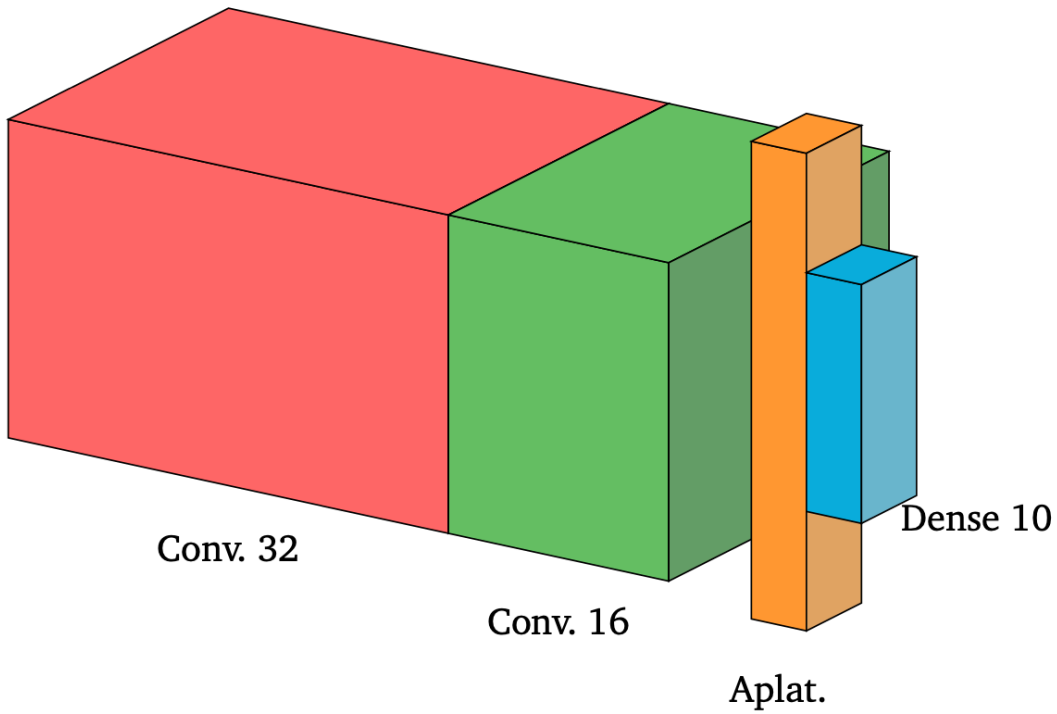


Figure – Source : exo7math

Algorithme des plus proches voisins

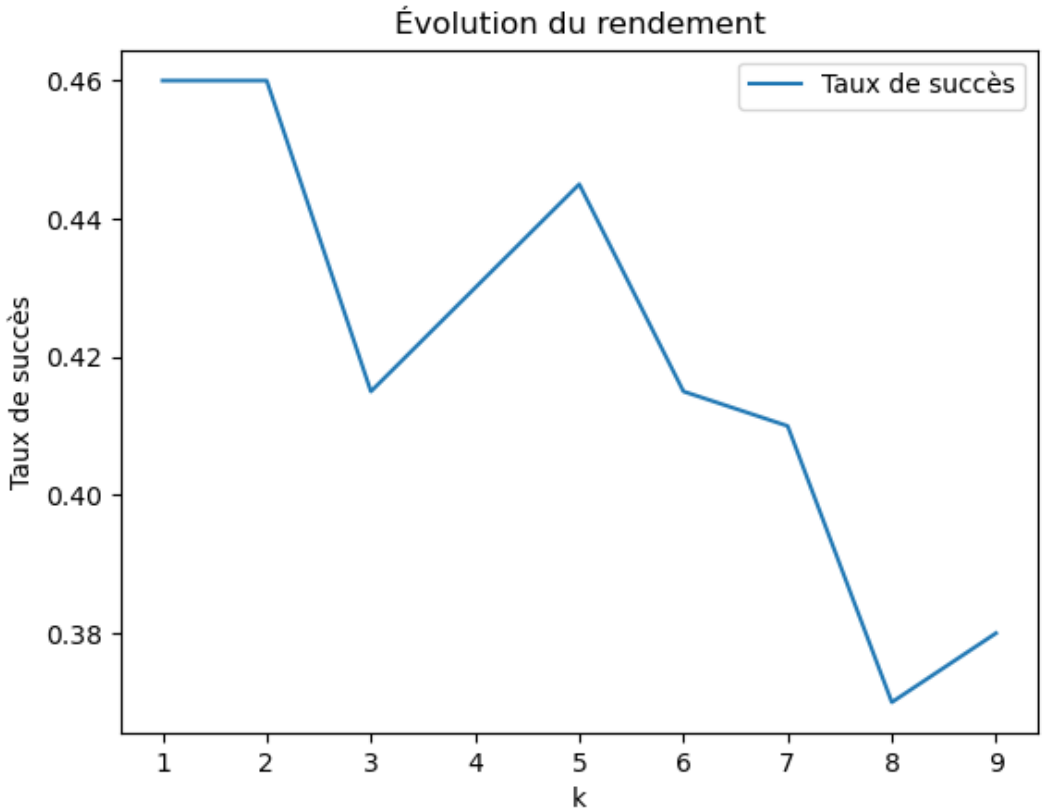


Figure – Résultats de l'algorithme des knn

Sommaire

Conclusion

- Résultats mitigés ;
- Difficultés et problèmes rencontrés :
  - Algorithmique complexe ;
  - Puissance de calcul ;
  - Données nécessaires.

FIN

Merci pour votre écoute.

Rétropropagation du perceptron

$$\frac{\partial C}{\partial w_{i,j}^{(n)}} = \frac{\partial C}{\partial v_j^{(n)}} \cdot \frac{\partial v_j^{(n)}}{\partial w_{i,j}^{(n)}}$$

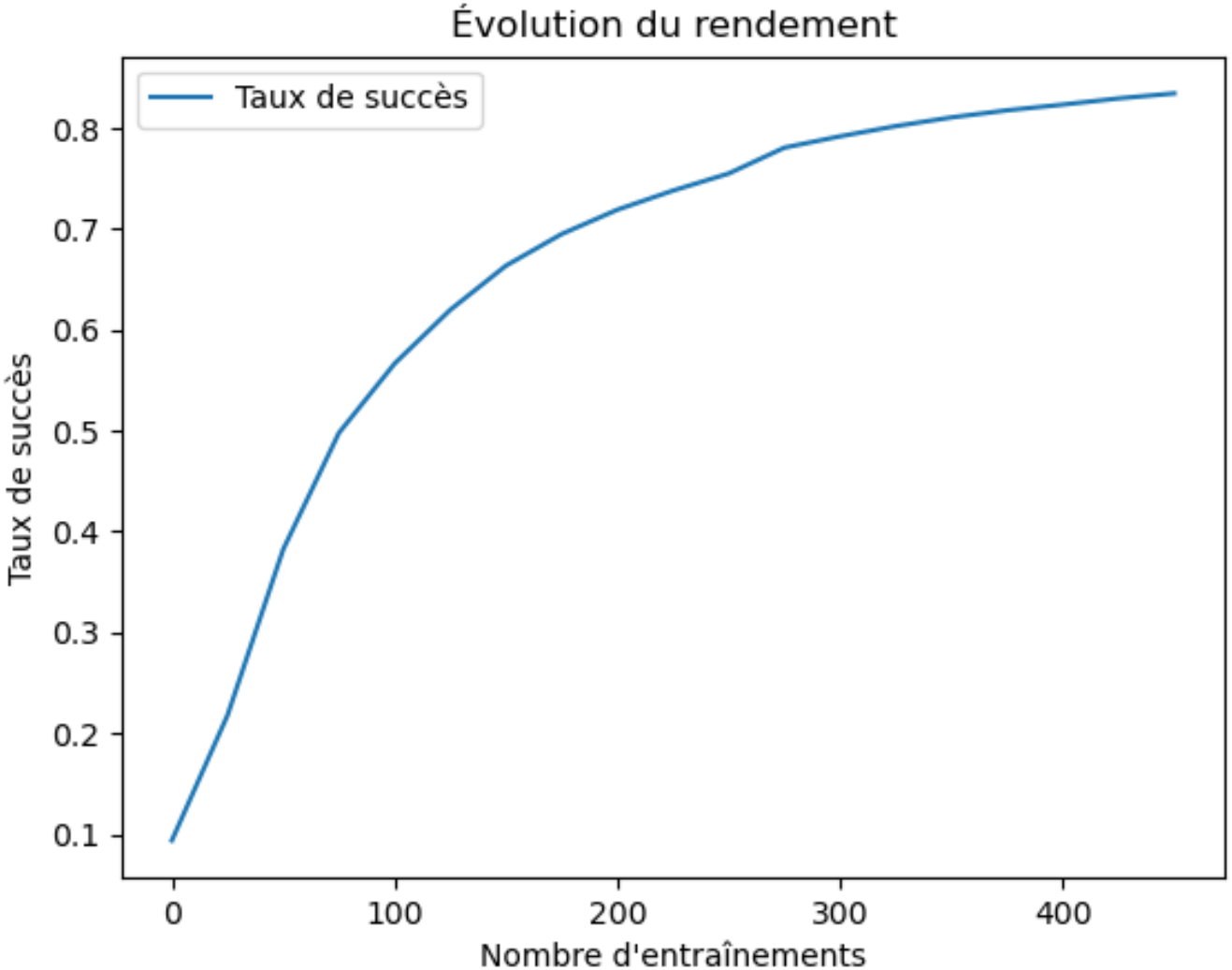
$$\frac{\partial v_j^{(n)}}{\partial w_{i,j}^{(n)}} = f_a' \left(\sum_{p=1}^{c_{k-1}} w_{p,j}^{(n)} \cdot v_p^{(n-1)} + b_j^{(n)} \right) \cdot v_i^{(n-1)}$$

$$\frac{\partial C}{\partial w_{i,j}^{(n)}} = \frac{\partial C}{\partial v_j^{(n)}} \cdot f_a' \left(\sum_{p=1}^{c_{k-1}} w_{p,j}^{(n)} \cdot v_p^{(n-1)} + b_j^{(n)} \right) \cdot v_i^{(n-1)}$$

$$\frac{\partial C}{\partial b_j^{(n)}} = \frac{\partial C}{\partial v_j^{(n)}} \cdot \frac{\partial v_j^{(n)}}{\partial b_j^{(n)}}$$

$$\frac{\partial C}{\partial b_j^{(n)}} = \frac{\partial C}{\partial v_j^{(n)}} \cdot f_a' \left(\sum_{p=1}^{c_{k-1}} w_{p,j}^{(n)} \cdot v_p^{(n-1)} + b_j^{(n)} \right)$$

MNIST


Évolution du rendement
Taux de succès
Taux de succès
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
100
200
300
400
Nombre d'entraînements

Code - Perceptron multicouche I

```python
import numpy as np
from annexe import *


class Reseau:
    def __init__(self, dimensions, taux_apprentissage, fonctions_activation_noms):
        print("\nDEFINITION DU RESEAU VECOTIRISE")
        self.informations_reseau = dimensions
        self.taux_apprentissage = taux_apprentissage
        self.fonctions_activation = fonctions_activation_noms
        self.A, self.V = [], []
        self.W, self.B = self.definition_reseau()
        self.nb_classes = self.informations_reseau[-1]
        self.nb_entrainements, self.nb_tests = 0, 0
        self.taux_reussite = []

    def definition_reseau(self):
        w, b = [], []
        for i in range(len(self.informations_reseau) - 1):
            w.append(np.random.randn(self.informations_reseau[i + 1],
            ↪  self.informations_reseau[i]) / 4)
            b.append(np.random.randn(self.informations_reseau[i + 1], 1) / 4)
        return w, b

    def propagation(self, x):
        self.A = [x]
        self.V = [x]
```

Code - Perceptron multicouche II

```
27          for i in range(len(self.informations_reseau) - 1):
28              self.A.append(np.dot(self.W[i], self.V[i]) + self.B[i])
29              self.V.append(self.fonctions_activation[i]("", self.A[i + 1]))
30
31      def retropropagation(self, y, d):
32          for k in range(1, len(self.V)):
33              if k == 1:
34                  d_v = self.V[-1] - y.T
35              else:
36                  d_v = np.dot(self.W[-k + 1].T, d_v) * self.fonctions_activation[-k]("derivee",
                    ↪  self.V[-k])
37              d_w = 1 / d * np.dot(d_v, self.V[-k - 1].T)
38              d_b = 1 / d * np.sum(d_v, axis=1).reshape(self.B[-k].shape[0], 1)
39              self.W[-k] -= self.taux_apprentissage * d_w
40              self.B[-k] -= self.taux_apprentissage * d_b
41
42      def entrainement(self, donnees, nb_repetitions):
43          x_train, y_train = donnees
44          y_train2 = ys_matriciels(y_train, self.nb_classes)
45          x_train = x_train.reshape(x_train.shape[0], x_train.shape[1] * x_train.shape[2]).T
46          self.nb_entrainements += x_train.shape[0]*nb_repetitions
47          print("\nENTRAINEMENT")
48          for i in range(nb_repetitions):
49              avance = i / nb_repetitions * 100
50              self.propagation(x_train)
51              self.retropropagation(y_train2, x_train.shape[1])
```

Code - Perceptron multicouche III

```
52              if avance % 5 == 0:
53                  resultats_corrects = comptage_resultats(np.argmax(self.V[-1], 0), y_train) /
                    ↪  y_train.shape[0]
54                  self.taux_reussite.append(resultats_corrects)
55                  print(int(avance), " % : rendement ", int(resultats_corrects * 100))
56
57      def test(self, donnees):
58          print("\nTEST")
59          x_test, y_test = donnees
60          x_test = x_test.reshape(x_test.shape[0], x_test.shape[1] * x_test.shape[2])
61          nombre_succes, nombre_donnees_test = 0, len(x_test)
62          for i in range(nombre_donnees_test):
63              avancee = i / nombre_donnees_test * 100
64              self.propagation(x_test[i].reshape(x_test[i].shape[0], 1))
65              valeur_pratique = np.argmax(self.V[-1])
66              if avancee % 10 == 0:
67                  print(avancee, " % : ")
68              if valeur_pratique == y_test[i]:
69                  nombre_succes += 1
70          print("\nNombre de succès : ", nombre_succes)
71          print("Taux de réussite : ", self.taux_reussite * 100, "%")
```

Code - MNIST I

```
1   from ReseauVectorise import Reseau
2   from annexe import ReLu, softmax, affichage, ys_matriciels
3   from matplotlib import pyplot as plt
4   import numpy as np
5   import keras
6
7
8   (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
9   x_train = x_train.astype("float32") / 255
10  x_test = x_test.astype("float32") / 255
11
12  reseau_mnist = Reseau([28*28, 10, 10], 0.10, [ReLu, softmax])
13  reseau_mnist.entrainement((x_train, y_train), 500)
14  reseau_mnist.test((x_test, y_test))
15
16  taux_succes = reseau_mnist.taux_reussite
17
18  affichage([0.05*i*500 for i in range(len(taux_succes))], [taux_succes], ["Taux de succès"],
    ↪  "Évolution du rendement", "Nombre d'entraînements", "Taux de succès")
```

Code - Convolution I

```python
import numpy as np
from annexe import lineaire


def convolution_2d(matrice, taille=3, fonction_activation=lineaire):
    n, p = matrice.shape
    h = taille // 2
    W = np.flip(np.random.randn(taille, taille))
    matrice_3 = np.zeros((n + 2 * h, p + 2 * h))
    matrice_3[h:-h, h:-h] = matrice
    matrice_2 = np.zeros((n, p))
    for i in range(n):
        for j in range(p):
            coucou = matrice_3[i-h+1:i+taille-h+1, j-h+1:j+taille-h+1]
            matrice_2[i, j] = fonction_activation("", np.sum(coucou * W))
    return matrice_2

def convolution_classiques(matrice, type):
    n, p = matrice.shape
    h = 1
    filtre = np.zeros((3,3))
    if type == "moyenne":
        filtre = 1/9 * np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
    elif type == "gaussien":
        filtre = 1/16 * np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])
    elif type == "pique":
        filtre = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
```

Code - Convolution II

```
28          elif type == "bords":
29              filtre = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
30          elif type == "relief":
31              filtre = np.array([[-2, -1, 0], [-1, 1, 1], [0, 1, 2]])
32          matrice_3 = np.zeros((n + 2 * h, p + 2 * h))
33          matrice_3[h:-h, h:-h] = matrice
34          matrice_2 = np.zeros((n, p))
35          for i in range(n):
36              for j in range(p):
37                  coucou = matrice_3[i-h+1:i+3-h+1, j-h+1:j+3-h+1]
38                  matrice_2[i, j] = np.sum(coucou * np.flip(filtre))
39          return matrice_2
```

Code - Réseau convolutif I

```
1   import numpy as np
2   from annexe import ys_matriciels, comptage_resultats
3   from scipy.signal import convolve2d
4
5
6   class ReseauConvolutif:
7       def __init__(self, nb_classes, dimensions, taux_apprentissage, nb_rep_entrainement,
        ↪  donnees_entrainement):
8           print("\nDEFINITION DU RÉSEAU")
9
10          self.nb_classes = nb_classes
11          self.informations_reseau = dimensions
12          self.fonctions_activation = []
13
14          self.taille_convolution = 2
15          self.taille_dense = 3
16
17          self.taux_apprentissage = taux_apprentissage
18          self.nb_rep_entrainement = nb_rep_entrainement
19          self.donnees_entrainement = donnees_entrainement
20
21          self.A, self.V = [], []
22          self.W, self.B, self.filtres, self.b2 =
           ↪  self.definition_reseau(donnees_entrainement[0].shape[1])
23
24          self.nb_entrainements, self.nb_tests = 0, 0
25          self.taux_reussite = []
```

Code - Réseau convolutif II

```
26          print(self.fonctions_activation)
27
28      def definition_reseau(self, hauteur_image):
29          w, b, f, b2 = [], [], [], []
30          for k in range(len(self.informations_reseau)-1):
31              if self.informations_reseau[k][0] == "C":
32                  f.append(np.random.randn(3, 3))
33                  b2.append(np.random.randn())
34                  self.fonctions_activation.append(self.informations_reseau[k][1])
35              else:
36                  if self.informations_reseau[k-1][0] == "C":
37                      w.append(np.random.randn(self.nb_classes, (hauteur_image-2*len(f))**2))
38                      b.append(np.random.randn(self.nb_classes, 1))
39                      self.fonctions_activation.append(self.informations_reseau[k][1])
40                  elif k == len(self.informations_reseau)-1:
41                      w.append(np.random.randn(self.nb_classes, self.nb_classes))
42                      b.append(np.random.randn(self.nb_classes, 1))
43                  else:
44                      w.append(np.random.randn(self.nb_classes, self.nb_classes))
45                      b.append(np.random.randn(self.nb_classes, 1))
46                      self.fonctions_activation.append(self.informations_reseau[k][1])
47
48          for i in b2:
49              print(i.shape)
50          return w, b, f, b2
51
```

Code - Réseau convolutif III

```
52      def propagation(self, x):
53          self.A = [x]
54          self.V = [x]
55          for k in range(len(self.filtres)):
56              self.A.append(convolve2d(self.V[-1], self.filtres[k], mode='valid') + self.b2[k])
57              self.V.append(self.fonctions_activation[k]("", self.A[-1]))
58
59          self.V[-1] = self.V[-1].reshape(self.V[-1].shape[0] * self.V[-1].shape[1], 1)
60
61          for i in range(len(self.informations_reseau) - len(self.filtres) - 1):
62              self.A.append(np.dot(self.W[i], self.V[-1]) + self.B[i])
63              self.V.append(self.fonctions_activation[i + len(self.filtres)]("", self.A[-1]))
64
65      def retropropagation(self, y):
66          y = y.T
67
68          for k in range(1, self.taille_dense):
69              if k == 1:
70                  d_v = self.V[-1] - y
71              else:
72                  d_v = np.dot(self.W[-k + 1].T, d_v) * self.fonctions_activation[-k]("derivee",
                    ↪  self.A[-k])
73              d_w = np.dot(d_v, self.V[-k - 1].T)
74              d_b = np.sum(d_v, axis=1).reshape(self.B[-k].shape[0], 1)
75
76              self.W[-k] -= self.taux_apprentissage * d_w
```

Code - Réseau convolutif IV

```
77              self.B[-k] -= self.taux_apprentissage * d_b
78
79          d_v = np.dot(self.W[-self.taille_dense + 1].T, d_v)
80          self.V[-k] = self.V[-k].reshape(int(self.V[-k].shape[0] ** .5), int(self.V[-k].shape[0]
            ↪  ** .5))
81          d_v = d_v.reshape(self.V[-k].shape[0], self.V[-k].shape[1])
82
83          for k in range(self.taille_dense, self.taille_convolution + self.taille_dense):
84              i = k - self.taille_dense
85
86              d_f = convolve2d(self.V[-k - 1], d_v, mode='valid')
87              d_b2 = np.sum(d_v)
88              d_v = convolve2d(d_v, np.flip(self.filtres[-i]), mode='full') *
                ↪  self.fonctions_activation[-k]("derivee", self.A[-k])
89
90              self.filtres[-i] = self.filtres[-i] - self.taux_apprentissage * d_f
91              self.b2[-(i + 1)] = self.b2[-(i + 1)] - self.taux_apprentissage * d_b2
92
93      def entrainement(self):
94          x_train, y_train = self.donnees_entrainement
95          self.nb_entrainements += x_train.shape[0] * self.nb_rep_entrainement
96
97          print("\nENTRAINEMENT")
98
99          for i in range(self.nb_rep_entrainement):
100             avance = i / self.nb_rep_entrainement * 100
```

Code - Réseau convolutif V

```
101             corr = 0
102
103             for j in range(len(x_train)):
104                 x = x_train[j]
105                 y = np.array([y_train[j]])
106                 self.propagation(x)
107                 self.retropropagation(ys_matriciels(y, self.nb_classes))
108
109                 corr += comptage_resultats(np.argmax(self.V[-1], 0) + 1, y)
110
111             if avance % 5 == 0:
112                 resultat = corr / len(x_train)
113                 self.taux_reussite.append(resultat)
114                 print(int(avance), " % : rendement ", int(resultat * 100))
115
116     def test(self, donnees):
117         print("\nTEST")
118
119         x_test, y_test = donnees
120         x_test = x_test.reshape(x_test.shape[0], x_test.shape[1] * x_test.shape[2])
121
122         nombre_succes, nombre_donnees_test = 0, len(x_test)
123
124         for i in range(nombre_donnees_test):
125             avancee = i / nombre_donnees_test * 100
126             self.propagation(x_test[i].reshape(x_test[i].shape[0], 1))
```

Code - Réseau convolutif VI

```
127             valeur_pratique = np.argmax(self.V[-1])
128             if avancee % 10 == 0:
129                 print(avancee, " % : ")
130             if valeur_pratique == y_test[i]:
131                 nombre_succes += 1
132
133         self.taux_reussite = nombre_succes / nombre_donnees_test
134
135         print("\nNombre de succès : ", nombre_succes)
136         print("Taux de réussite : ", self.taux_reussite * 100, "%")
```

Code - Données I

```
1   import numpy as np
2   import pandas as pd
3   from PIL import Image
4
5
6   valeurs = pd.read_csv('data.csv', index_col=0)
7   valeurs = valeurs.to_numpy().reshape(1200)
8   images = np.zeros((1200, 32, 32))
9
10  for i in range(1200):
11      if i < 10:
12          k = '000'+str(i)
13      elif i < 100:
14          k = '00'+str(i)
15      elif i < 1000:
16          k = '0'+str(i)
17      else:
18          k = str(i)
19      images[i] = np.array(Image.open('numbers/' + k + '.png'))
```

Code - Perceptron I

```python
from ReseauVectorise import Reseau
from MaillotsData import images, valeurs
from annexe import ReLu, softmax, affichage, ys_matriciels
from convolution import convolution_classiques

images = images.astype("float32") / 255
liste = ["moyenne", "gaussien", "pique", "bords", "relief"]
y, legendes = [], []

for k in range(len(liste)):
    for i in range(len(images)):
        images[i] = convolution_classiques(images[i], liste[k])

    x_train, y_train = images[:1000], valeurs[:1000]
    x_test, y_test = images[1000:], valeurs[1000:]

    reseau = Reseau([32*32, 45, 45], 0.01, [ReLu, softmax])
    reseau.entrainement((x_train, y_train), 2000)
    taux_succes = reseau.taux_reussite
    legendes.append("Taux pour "+liste[k])
    y.append(taux_succes)

affichage([0.05*i*2000 for i in range(len(y[0]))], y, legendes, "Évolution du rendement",
↪  "Nombre d'entraînements", "Taux de succès")
```

Code - CNN 1 I

```python
from ReseauConvolution import ReseauConvolutif
from MaillotsData import images, valeurs
from annexe import ReLu, softmax, lineaire, ys_matriciels


x_train, y_train = images[:1000], valeurs[:1000]
x_test, y_test = images[1000:], valeurs[1000:]

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

reseau = ReseauConvolutif(45, [
    ("C", ReLu),
    ("C", lineaire),
    ("D", ReLu),
    ("D", softmax),
    ("D", lineaire)
], 0.01, 1000, (x_train, y_train))

reseau.entrainement()
```

Code - CNN 1 II

```
1   import numpy as np
2   from scipy.signal import convolve2d
3   from annexe import ReLu, softmax, ys_matriciels, tanh, affichage
4   from MaillotsData import images, valeurs
5
6   n = 1000
7   x_train, y_train = images[:n], valeurs[:n]
8   x_test, y_test = images[n:], valeurs[n:]
9   x_train = x_train.astype("float32") / 255
10  x_test = x_test.astype("float32") / 255
11
12  # Définitions générales
13  nb_train = len(x_train)
14  nb_entrainement = 1000
15  taux = 0.1
16  F1 = np.random.randn(3, 3)
17  B1 = np.random.randn()
18  F2 = np.random.randn(3, 3)
19  B2 = np.random.randn()
20  W3 = np.random.randn(45, 28 * 28)
21  B3 = np.random.randn(45, 1)
22  W4 = np.random.randn(45, 45)
23  B4 = np.random.randn(45, 1)
24
25  succes = []
26  for j in range(nb_entrainement):
```

Code - CNN 1 III

```
27          d = {}
28          compteur = 0
29          for i in range(nb_train):
30              # Propagation
31              V0 = x_train[i]
32              A1 = convolve2d(V0, F1, mode='valid') + B1
33              V1 = ReLu("", A1)
34              V2 = convolve2d(V1, F2, mode='valid') + B2
35              V22 = V2.reshape(28 * 28, 1)
36              A3 = np.dot(W3, V22) + B3
37              V3 = ReLu("", A3)
38              A4 = np.dot(W4, V3) + B4
39              V4 = softmax("", A4)
40
41              y_t = np.argmax(V4)
42
43              if y_t == y_train[i]:
44                  compteur += 1
45                  if y_t in d:
46                      d[y_t] += 1
47                  else:
48                      d[y_t] = 1
49
50              # Rétropopagation
51              y = ys_matriciels([y_train[i]], 45).T
52
```

Code - CNN 1 IV

```
53            dV4 = V4 - y
54            dW4 = np.dot(dV4, V3.T)
55            dB4 = dV4
56            W4 -= taux * dW4
57            B4 -= taux * dB4
58
59            dV3 = np.dot(W4.T, dV4) * ReLu("derivee", A3)
60            dW3 = np.dot(dV3, V22.T)
61            dB3 = dV3
62            W3 -= taux * dW3
63            B3 -= taux * dB3
64
65            dV2 = np.dot(W3.T, dV3)
66            dV2 = dV2.reshape(28, 28)
67            dF2 = convolve2d(V1, dV2, mode='valid')
68            dB2 = np.sum(dV2)
69            F2 -= taux * dF2
70            B2 -= taux * dB2
71
72            dV1 = convolve2d(dV2, np.flip(F2), mode='full') * ReLu("derivee", A1)
73            dF1 = convolve2d(V0, dV1, mode='valid')
74            dB1 = np.sum(dV1)
75            F1 -= taux * dF1
76            B1 -= taux * dB1
77
78        print(d)
```

Code - CNN 1 V

```
79          succes.append(compteur / nb_train)
80
81      affichage([i for i in range(len(succes))], [succes], ["nombre de succès"], "nombre de succès par
        ↪  itération",
82                "nombre de succès", "itération")
```

Code - CNN 2 I

```
1   import numpy as np
2   import keras
3   from keras import layers
4   from MaillotsData import images, valeurs
5
6   nb_classes = 46
7   taille = (32, 32, 1)
8   x_train, y_train = images[:1000], valeurs[:1000]
9   x_test, y_test = images[1000:], valeurs[1000:]
10  x_train = x_train.astype("float32") / 255
11  x_test = x_test.astype("float32") / 255
12  x_train = np.expand_dims(x_train, -1)
13  x_test = np.expand_dims(x_test, -1)
14  print("x_train shape:", x_train.shape)
15  print(x_train.shape[0], "train samples")
16  print(x_test.shape[0], "test samples")
17
18  y_train = keras.utils.to_categorical(y_train, nb_classes)
19  y_test = keras.utils.to_categorical(y_test, nb_classes)
20
21  reseau = keras.Sequential(
22      [
23          keras.Input(shape=taille),
24          layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
25          layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
26          layers.Flatten(),
27          layers.Dense(nb_classes, activation="softmax"),
```

Code - CNN 2 II

```
28          ]
29      )
30
31      reseau.summary()
32      batch_size = 128
33      epochs = 1000
34      reseau.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
35      reseau.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
36
37      resultats = reseau.evaluate(x_test, y_test, verbose=0)
38      print("Taux de succès : ", resultats[1])
```

Code - Voisins I

```
1   import numpy as np
2   from matplotlib import pyplot as plt
3   from MaillotsData import images, valeurs
4   from annexe import affichage
5
6
7   x_train, y_train = images[:1000], valeurs[:1000]
8   x_test, y_test = images[1000:], valeurs[1000:]
9   x_train = x_train.astype("float32") / 255
10  x_test = x_test.astype("float32") / 255
11
12
13  def distance(p1, p2):
14      d = 0
15      for i in range(p1.shape[0]):
16          for j in range(p1.shape[1]):
17              d += (p1[i, j] - p2[i, j])**2
18      return d**0.5
19
20
21  def voisins(x, k):
22      indices = sorted(range(len(x_train)), key=lambda i: distance(x, x_train[i]))
23      return indices[:k]
24
25
26  def plus_frequent(liste):
27      compte = {}
```

Code - Voisins II

```
28          for e in liste:
29              compte[e] = compte.get(e, 0) + 1
30          return max(compte, key=compte.get)
31
32
33      def knn(x, k):
34          V = voisins(x, k)
35          return plus_frequent([y_train[i] for i in V])
36
37
38      def precision(k):
39          n = 0
40          for i in range(len(x_test)):
41              if knn(x_test[i], k) == y_test[i]:
42                  n += 1
43          return n / len(x_test)
44
45
46      ks = range(1, 10)
47      resultats = [precision(k) for k in ks]
48      affichage(ks, [resultats], ["Précision en fonction de k"], "k voisins", "Rendement de
    ↪  l'algorithme")
```

Code annexe I

```python
import matplotlib.pyplot as plt
import numpy as np


def lineaire(mode, matrice):
    if mode == "derivee":
        return np.zeros(matrice.shape) + 1
    return matrice


def sigmoid(mode, matrice):
    if mode == "derivee":
        return sigmoid("", matrice) * (1 - sigmoid("", matrice))
    return 1 / (1 + np.exp(-matrice))


def ReLu(mode, matrice):
    if mode == "derivee":
        return matrice > 0
    return np.maximum(matrice, 0)


def tanh(mode, matrice):
    if mode == "derivee":
        return 1 - tanh("", matrice) ** 2
    return np.tanh(matrice)

```

Code annexe II

```
28
29  def softmax(mode, matrice):
30      e = np.exp(matrice - np.max(matrice))
31      return e / sum(e)
32
33
34  def ys_matriciels (y, nb_classe):
35      return np.array([[1 if i == j else 0 for i in range(nb_classe)] for j in y])
36
37
38  def comptage_resultats(y_pratiques, y_theoriques):
39      compteur = 0
40      for i in range(y_pratiques.size):
41          if y_pratiques[i] == y_theoriques[i]:
42              compteur += 1
43      return compteur
44
45
46  def cout(x, y):
47      return 0.5 * np.sum((x-y)**2)
48
49
50  def image(image):
51      plt.imshow(image, cmap='gray')
52      plt.show()
53
```

Code annexe III

```
54
55  def image_resultat(image, valeurs, legendes):
56      maxi = np.argmax(valeurs)
57      couleurs = ["red" if i == maxi else "blue" for i in range(len(valeurs))]
58      plt.figure()
59      plt.subplot(211)
60      plt.imshow(image)
61      plt.subplot(212)
62      plt.bar(legendes, valeurs, color=couleurs)
63      plt.show()
64
65
66  def images_comparaison(image_originale, image_retouchee):
67      plt.figure()
68      plt.subplot(211)
69      plt.imshow(image_originale)
70      plt.subplot(212)
71      plt.imshow(image_retouchee)
72      plt.show()
73
74
75  def affichage(xs, ys, legendes, titre="", x_label="", y_label=""):
76      for i in range(len(ys)):
77          plt.plot(xs, ys[i], label=legendes[i])
78      plt.title(titre)
79      plt.xlabel(x_label)
```

Code annexe IV

```
80          plt.ylabel(y_label)
81          plt.legend()
82          plt.savefig("graphique.png")
83          plt.show()
```