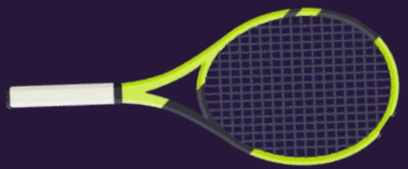
AUTOMATISATION DES DÉCISIONS PRISES DANS L'ARBITRAGE SPORTIF

(exemple du Tennis)

EL MAZROUA AHMED
N° d'inscription: 19634



MOTIVATION:

Figure 1 – Championnat européen universitaire de tennis (EUSA)

Figure 2 – Yannick Noah, légende du tennis français, célèbre pour sa victoire à Roland-Garros en 1983

MOTIVATION:

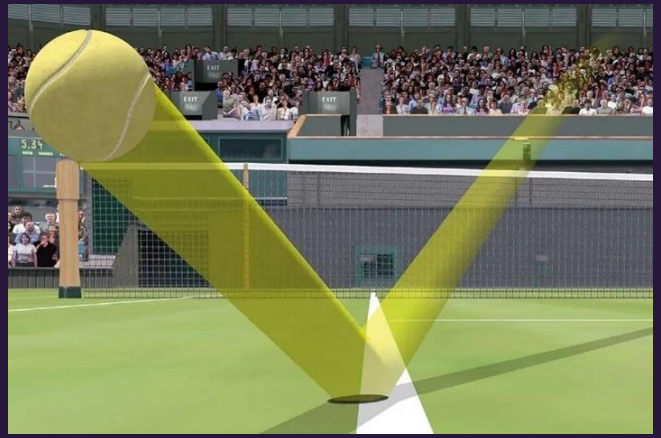


Figure 3 – Une balle de tennis rebondissant, réalisée par la technologie Hawk-Eye.

Figure 4 – L'une des caméras utilisées en Hawk-Eye.

Contraintes:

- Limité aux professionnels
- Coût d'équipement élevé **(entre 6 et 10 caméras)**

PROBLEMATIQUE:

Comment automatiser détermination du statut in/out de la balle du tennis, en minimisant le coût?

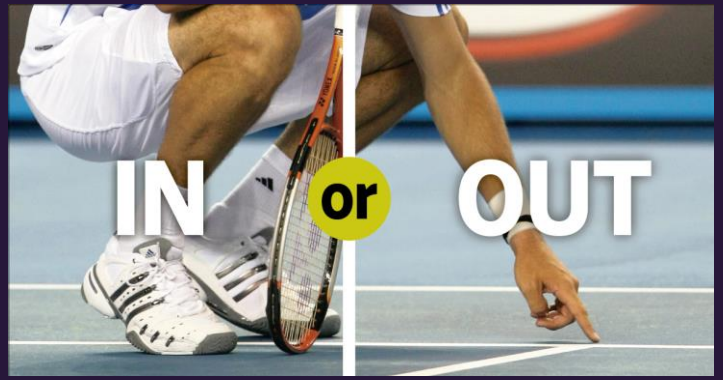
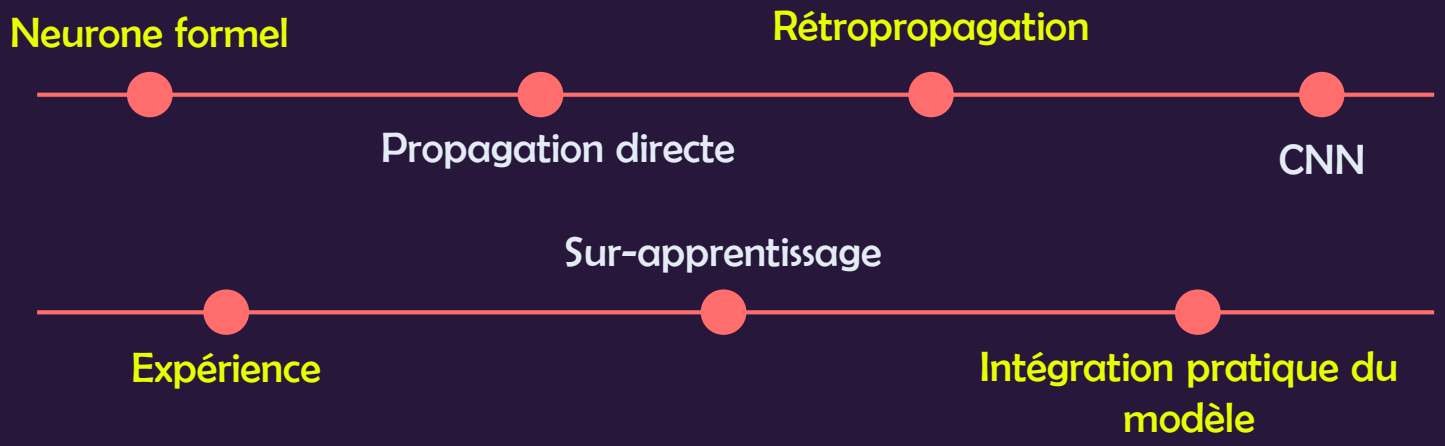


Figure 5 – « IN » ou « OUT » ?

PLAN:

- **Première approche:** Etude de la trajectoire
- **Deuxième approche:** Apprentissage supervisé



Première approche:

Objectif: construire la trajectoire de la balle => son point d'impact avec le sol

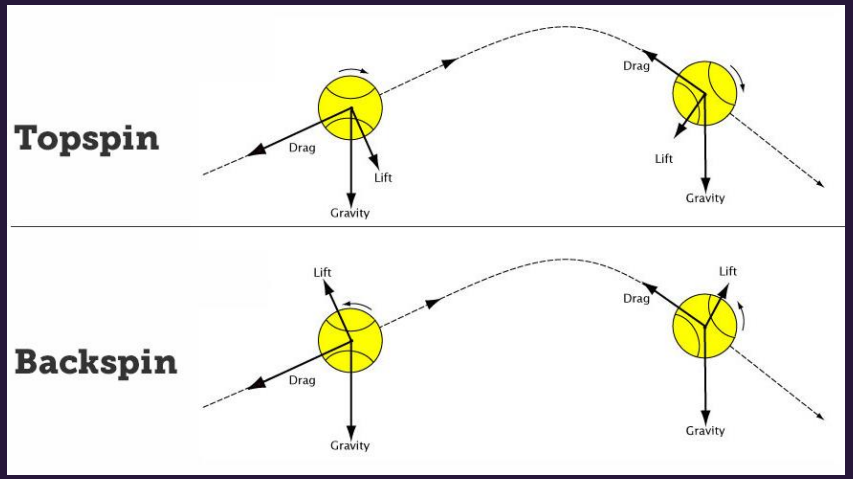


Figure 6 – trajectoire d'une balle de tennis avec deux types de spin, chacun influençant la direction des forces aérodynamiques

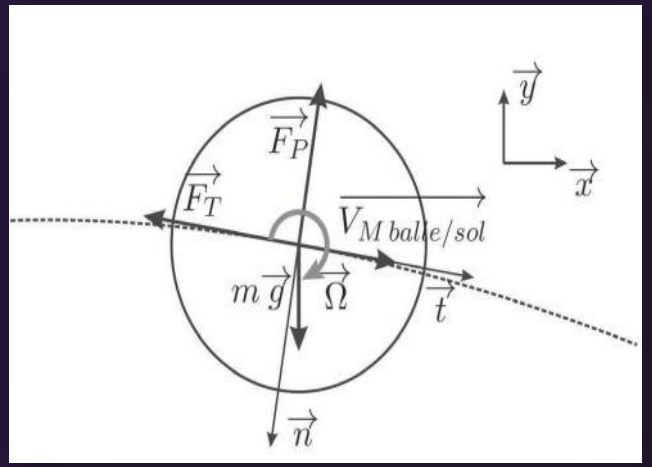


Figure 7 – Bilan des forces

Simulation:

Equation de mouvement:

Principe fondamentale de la dynamique: $m\frac{d^2\overrightarrow{OG}}{dx^2} = \overrightarrow{F_P} + \overrightarrow{F_T} + m\vec{g}$

On trouve donc par projection:

avec: $\overrightarrow{F_T} = -\frac{1}{2} C_d \rho_{air} S V^2 \vec{t}$

$\overrightarrow{F_P} = \mathrm{L}\,\overrightarrow{\Omega} \wedge \overrightarrow{V}$

$$m\frac{d^2x(t)}{dt^2} = -\frac{1}{2} C_d \rho_{air} S V^2 cos(\alpha) - L\,\Omega\, V sin(\alpha)$$

$$m\frac{d^2y(t)}{dt^2} = -mg - \frac{1}{2} C_d \rho_{air} S V^2 sin(\alpha) + L\,\Omega\, V cos(\alpha)$$

La résolution numérique permet de tracer la trajectoire:

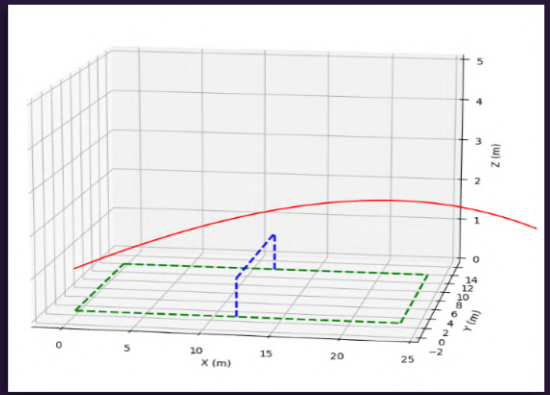


Figure 8 – Trajectoire de la balle

Expérience:

Figure 9 – Lanceur de ballon Slinger

Vitesse initiale: 30 m/s

Figure 10 – Balle en position initiale

Figure 11 – Balle en position finale

Evaluation:

L'écart entre la simulation et l'expérience est très significatif: **erreur de +-8cm**

Deuxième approche:

(apprentissage supervisé)

Définition: méthode d'apprentissage en machine où un modèle est entraîné à partir de données étiquetées **(Ball in - Ball out)**

Objectif: prédire les sorties correctes pour de nouvelles entrées basées sur ce qu'il a appris à partir des données d'entraînement.

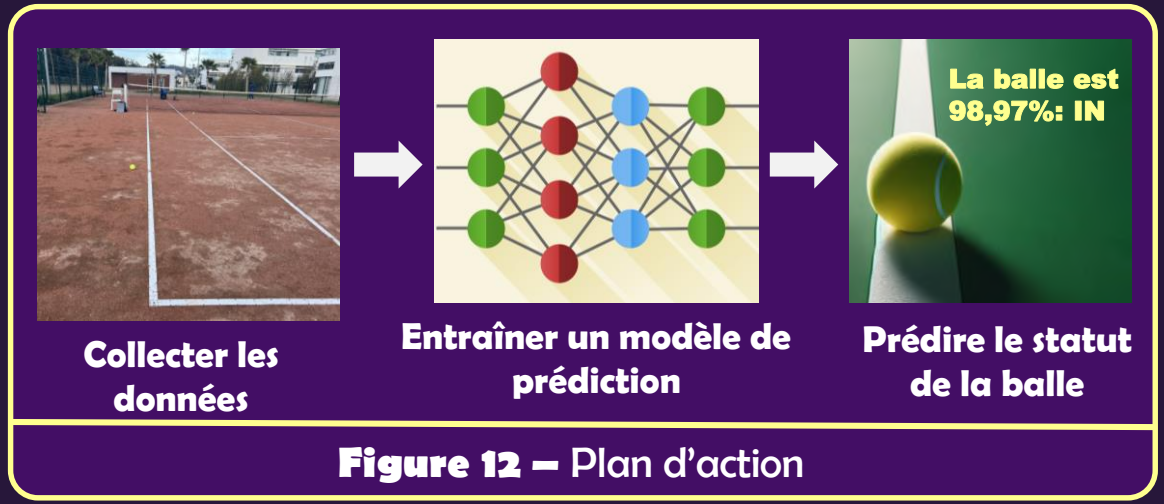
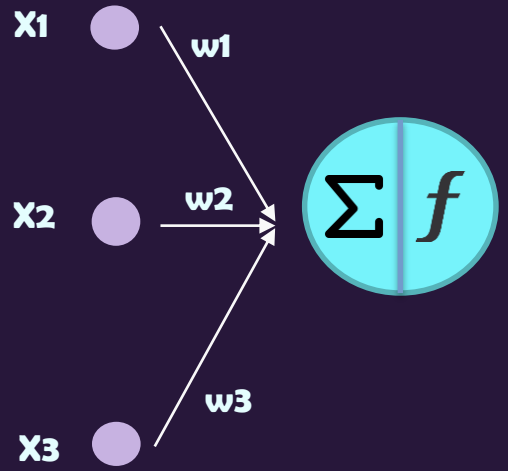


Figure 12 – Plan d'action

LE NEURONE FORMEL:



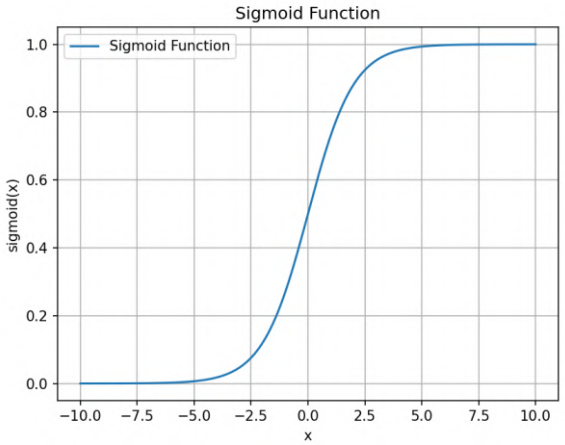
$Y = f(X1*w1+X2*w2+X3*w3 + b)$

Où: f est une fonction d'activation

FONCTION D'ACTIVATION:

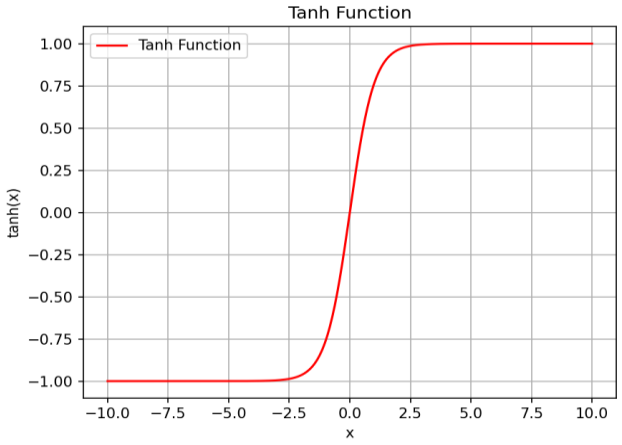
Sigmoïd:

$$f(x) = \frac{1}{1+e^{-x}}$$



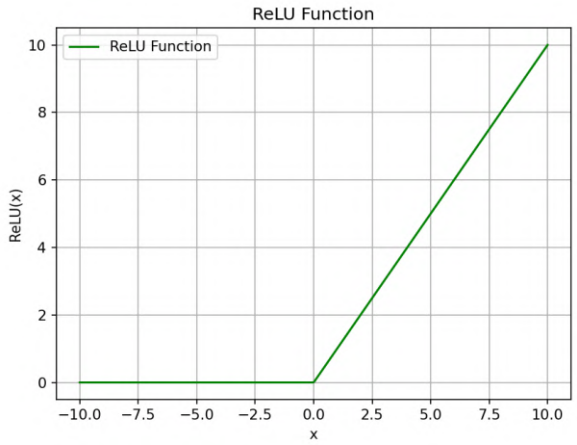
Tangente hyperbolique:

$$f(x) = \tanh(x)$$



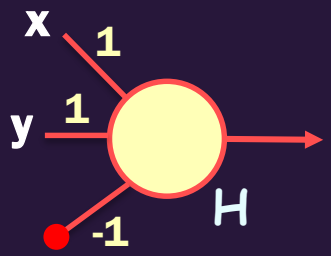
Relu:

$$f(x) = \max(0,x)$$

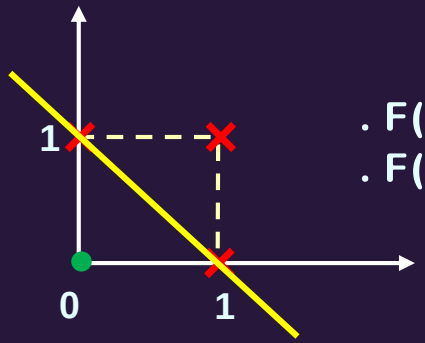


EXEMPLE:

- Un neurone qui modélise l'opérateur logique **«ou»** :

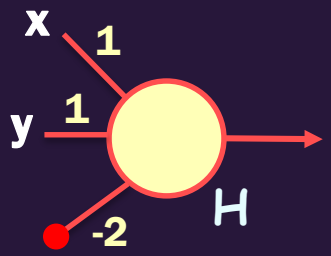


F(x,y)=H(x+y-1)

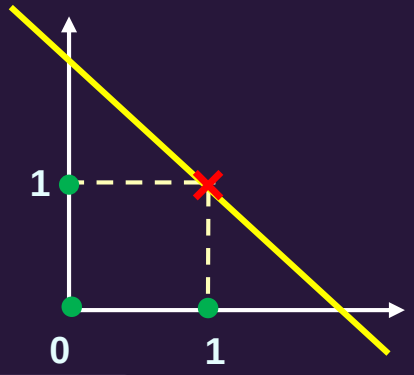


. F(1,0)=H(0)=1
. F(0,0)=H(-1)=0

- Un neurone qui modélise l'opérateur logique **«et»** :



F(x,y)=H(x+y-2)



. F(1,1)=H(0)=1
. F(1,0)=H(-1)=0

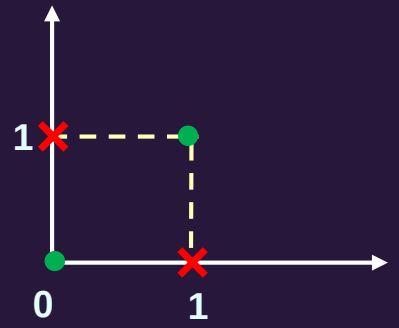
H: fonction d'activation Heaviside

$$H(x)=\begin{cases}0 \text{ si } x<0\\ 1 \text{ si } x>=1\end{cases}$$

MAIS, un neurone est-il suffisant?

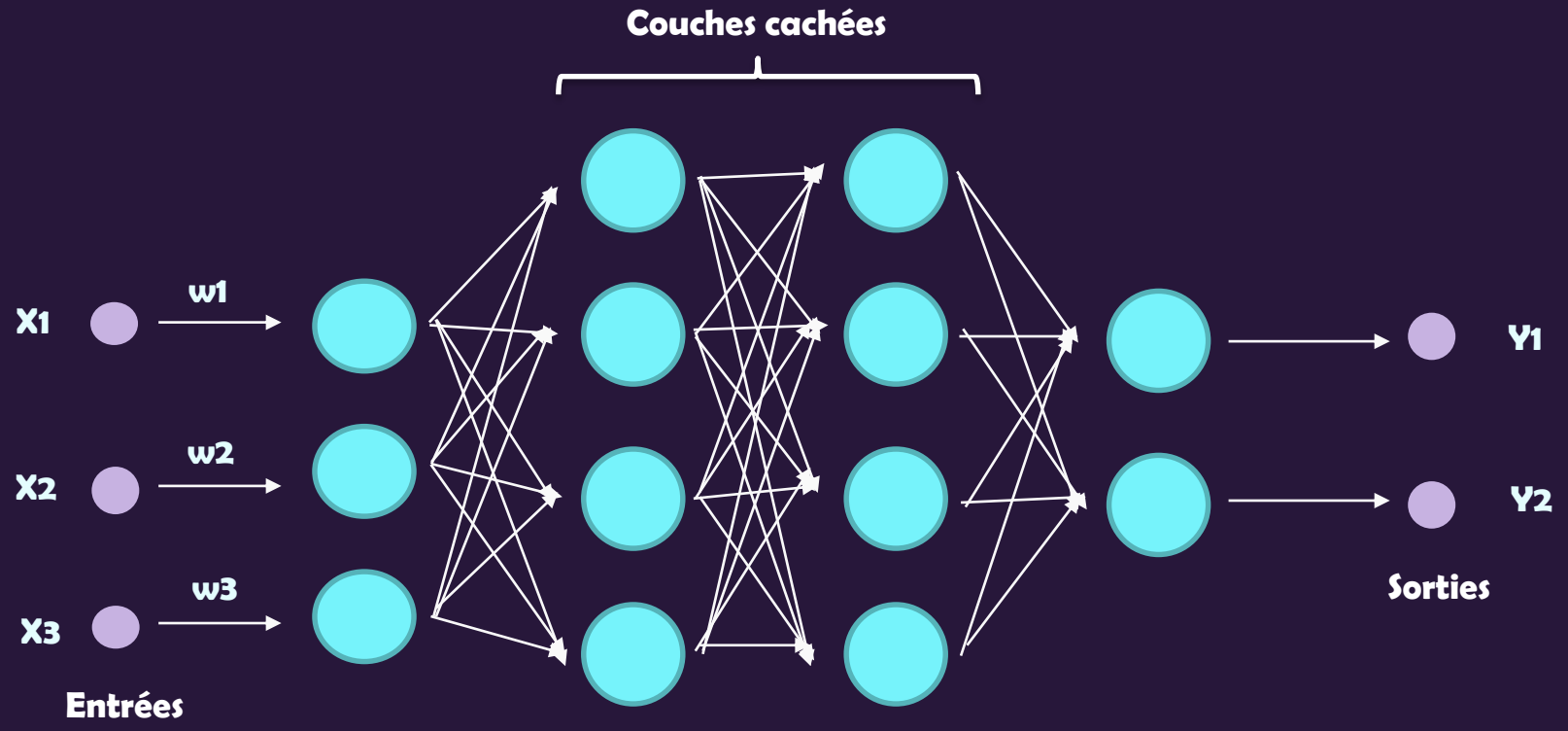
Non, car il n'existe pas de neurone qui modélise l'opérateur logique «Xor» ou «ou exclusif» (par exemple) :

X	Y	X xor Y
0	0	0
0	1	1
1	0	1
1	1	0

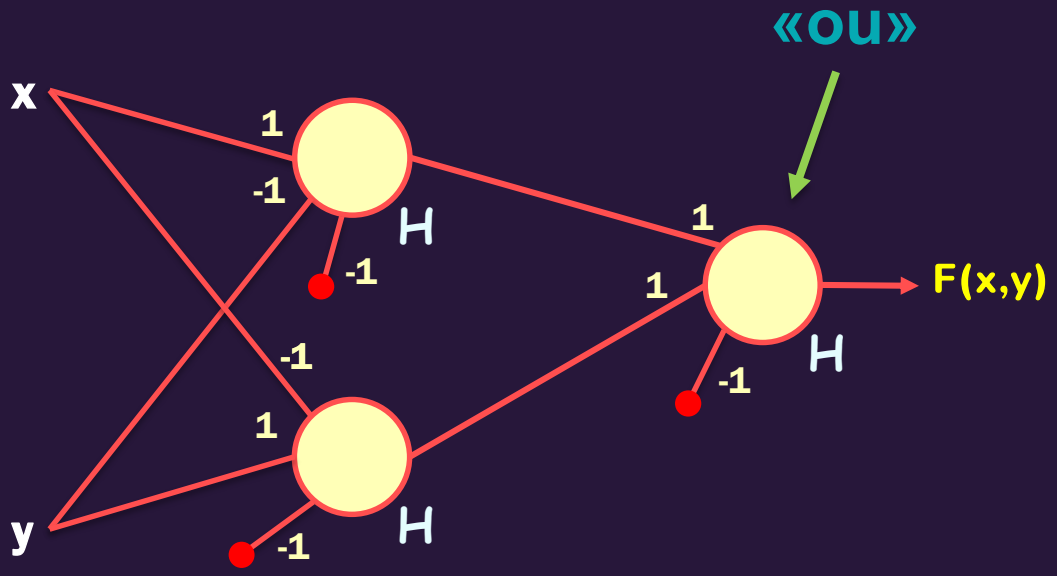


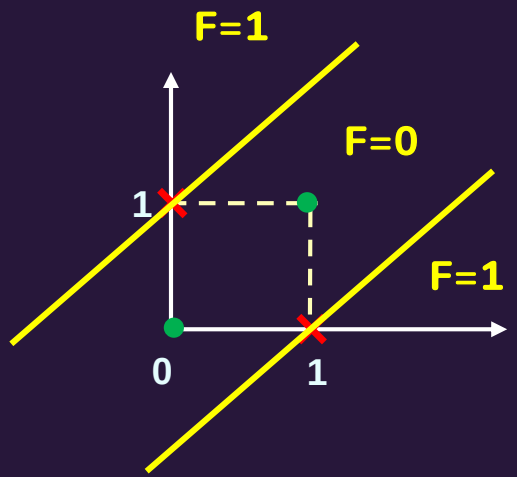
(Démonstration: Voir annexe)

PROPAGATION DIRECTE:



Un réseau représentant le «ou exclusif» devient possible :





RETROPROPAGATION:

Ecart de la prédiction: fonction de coût

Plus souvent l'erreur quadratique moyenne telle que:

$$RMSE(y_f,y) = \frac{1}{2n}\sum_i (yr_{éelle} - y_{prédite})^2$$

ou encore:

$$MAE(y_f,y) = \frac{1}{2n}\sum_i |y_{réelle} - y_{prédite}|$$

DESCENTE DE GRADIENT:

Objectif: Trouver $x^* \in \mathbb{R}^n$ tel que: $f(x^*)=\min_{x\in\mathbb{R}^n} f(x)$ où $f: \mathbb{R}^n \to \mathbb{R}$, de classe C^1 (au moins)

Figure 13 – Perdu en montagne, comment retrouver son chemin?

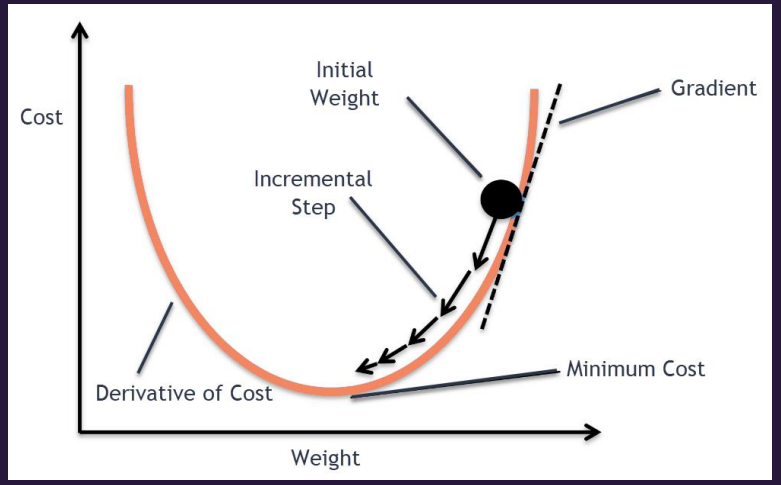


Figure 14 – Descente de gradient pour une fonction (à une seule variable)

DESCENTE DE GRADIENT:

Exemple: On considère la fonction $g: \mathbb{R}^2 \longrightarrow \mathbb{R}$

$$(x,y) \longmapsto x^4 + y^2$$

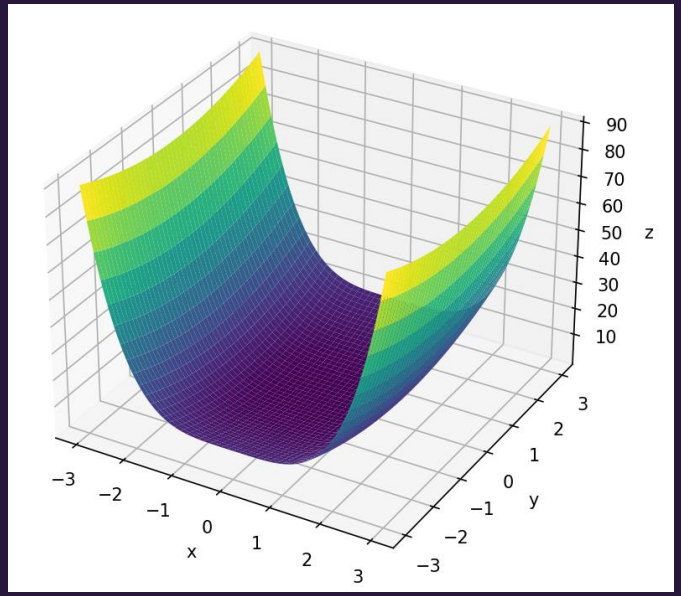
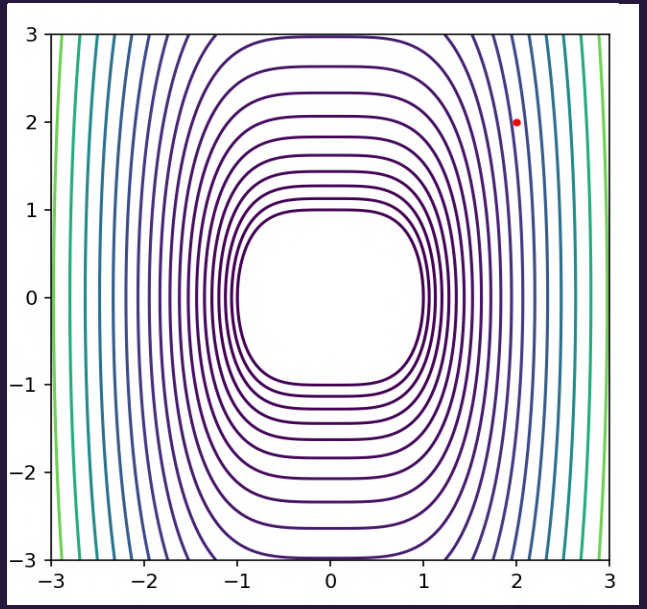
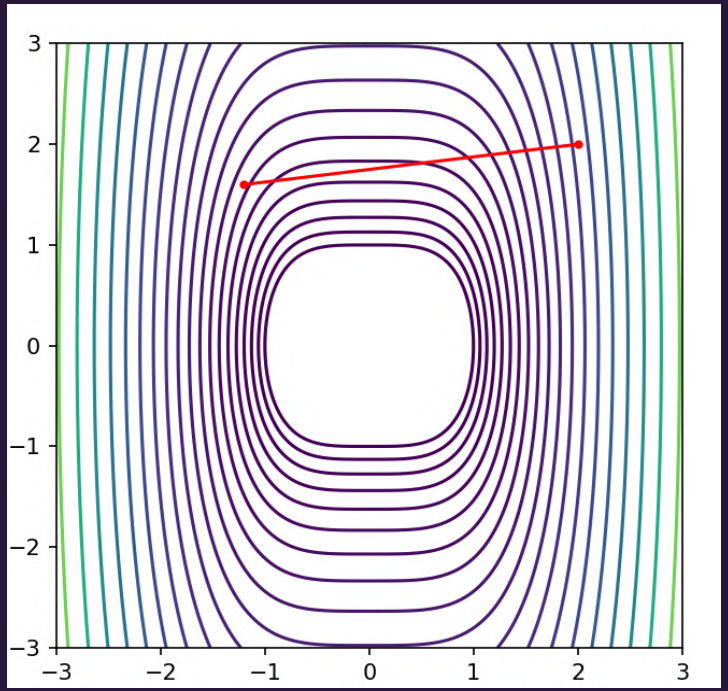


Figure 15 – Représentation de la fonction g en 3D

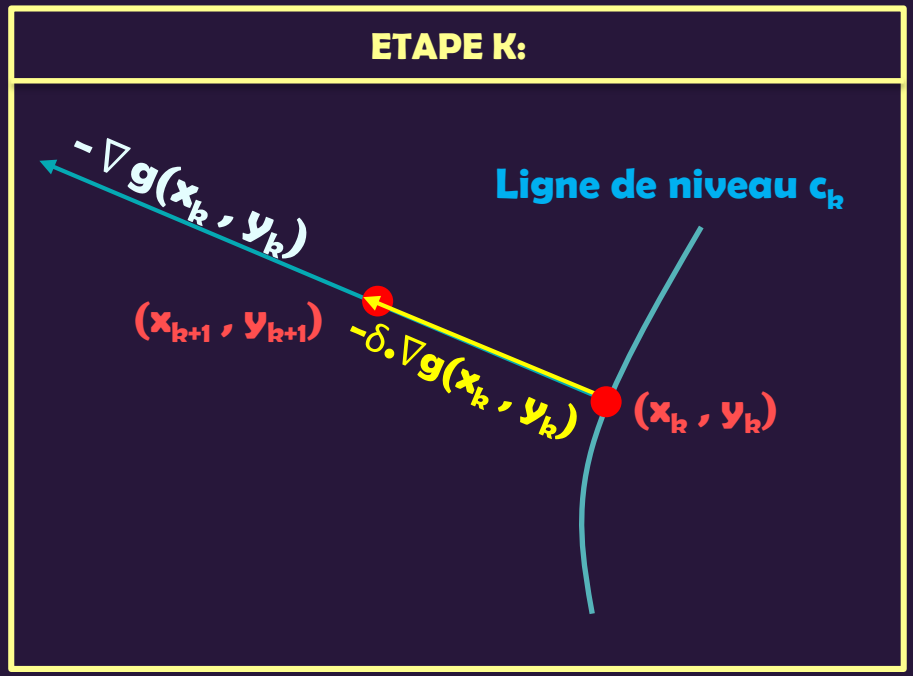


ETAPE 1: On prend le point (2,2) et on note c1=g((2,2))

DESCENTE DE GRADIENT:

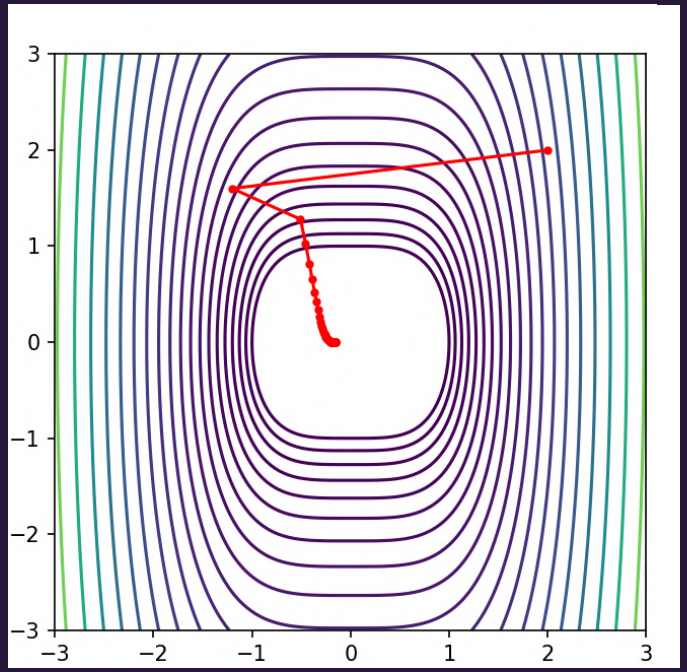


ETAPE 2: On calcule $\nabla g(x,y)$ en (2,2) et on se déplace dans la direction de $-\delta.\nabla g(2,2)$ (par exemple $\delta = 0.1$)



Par construction de δ , $c_{k+1} < c_k = g((x_k, y_k))$

DESCENTE DE GRADIENT:



ETAPE … : Après plusieurs itérations (ici 50), on voit la progression vers le minimum de la fonction g.

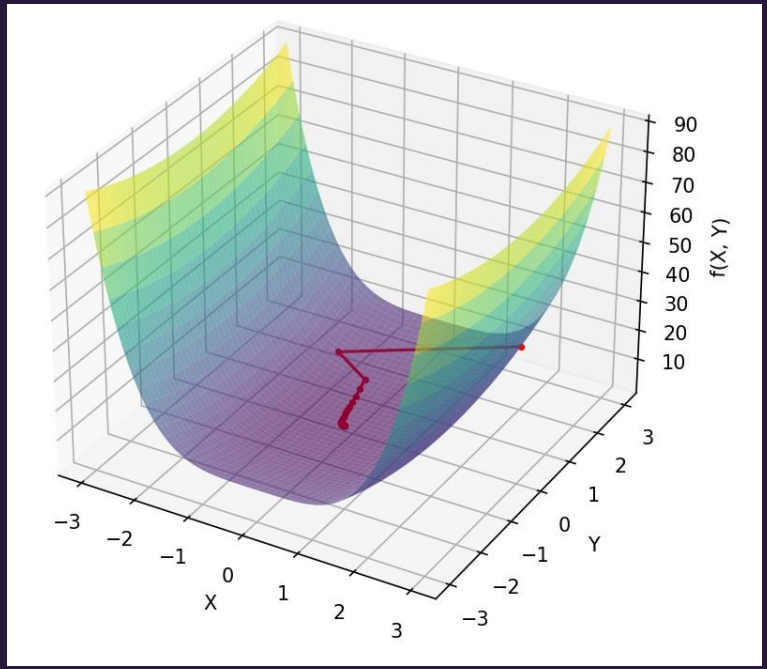


Figure 16 – Représentation de la fonction g en 3D, après les 50 itérations

DESCENTE DE GRADIENT:

Objectif: Minimiser la fonction de coût

La modification des paramètres se fait tel que :

$$\theta_{ij} := \theta_{ij} - \alpha \,.\, \nabla C$$

Le gradient se calcule avec la dérivée partielle de la fonction de coût par rapport à chacun des paramètres:

$$\nabla C_x(\theta) = \left(\frac{\partial C}{\partial \theta_1} \quad \frac{\partial C}{\partial \theta_2} \quad \frac{\partial C}{\partial \theta_3} \quad \dots \right)^T$$

α : **un scalaire, représentant le taux d'apprentissage.**
∇C : **le gradient de la fonction de coût.**
θ: **poids ou biais**

Soit ∇_i le gradient sur un perceptron de la couche k:

$$J_k = \begin{pmatrix} \nabla_1 \\ \nabla_2 \\ \nabla_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial \theta_{1,1}} & \cdots & \frac{\partial C}{\partial \theta_{1,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial \theta_{m,1}} & \cdots & \frac{\partial C}{\partial \theta_{m,n}} \end{pmatrix} \quad \text{or:} \quad \nabla C = \frac{\partial C}{\partial \theta_{i,j}}$$

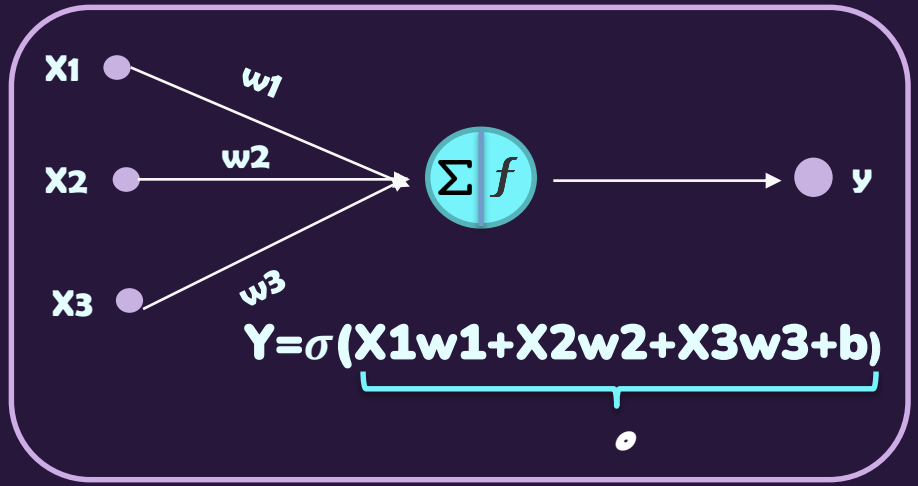
en utilisant la règle de chaîne, on calcule la perte où $\theta_{i,j}$ représente le poids $w_{i,j}$:

$$\nabla C = \frac{\partial C}{\partial w} = \frac{\partial o}{\partial w}\frac{\partial y}{\partial o}\frac{\partial C}{\partial y}$$

$$\frac{\partial C}{\partial y_i} = \frac{\partial}{\partial yi}\left(\frac{1}{2n}\sum_{i=0}^{n}(yi - yf)^2\right) = (yi - yf)$$

$$\frac{\partial y}{\partial o} = y(1-y)$$

$$\frac{\partial o}{\partial w_i} = x_i$$



L'apprentissage s'arrête lorsque les paramètres convergent vers des valeurs, et que la dérivée de la fonction de coût vaut 0 **(rétropropagation)**

RÉSEAU DE NEURONES CONVOLUTIONNEL (CNN):

Objectif: reconnaître des objets

Principe: glisser des filtres sur l'entrée pour identifier des motifs visuel

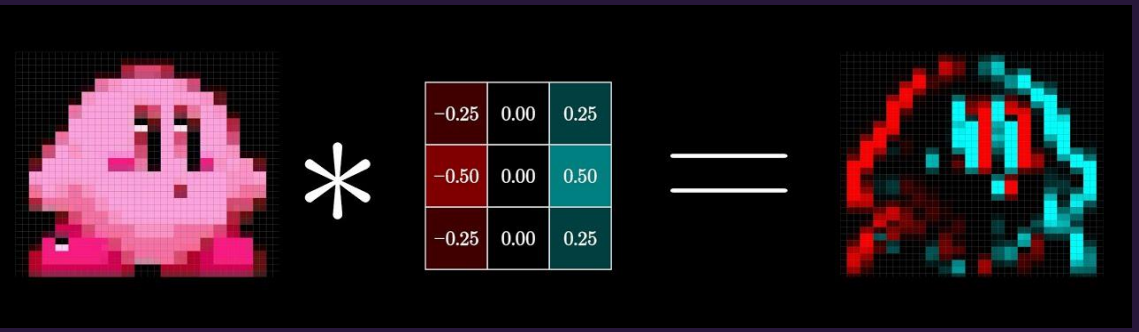


Figure 17 – Convolution 2D Image

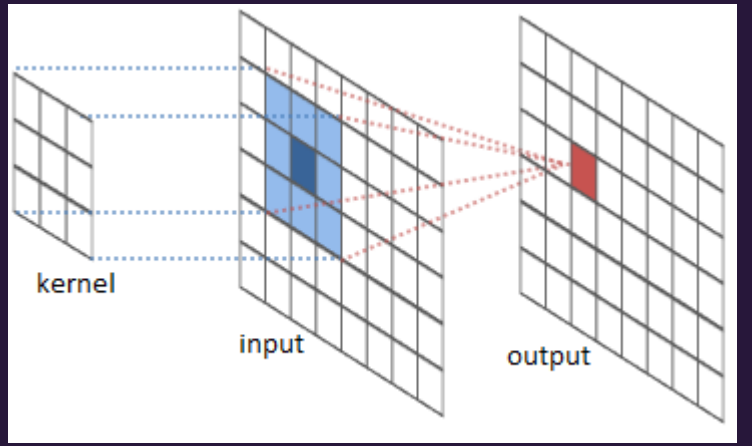


Figure 18 – Schéma du processus

EXEMPLES:
(en traitement d'images)

Flou:

$$* \frac{1}{9}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} =$$

Piqué:

$$* \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} =$$

Maxpooling:

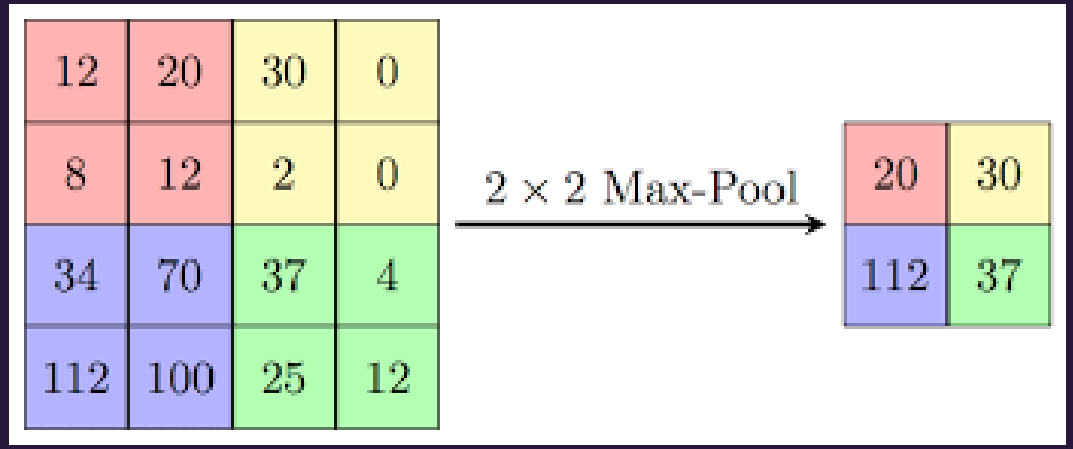


Figure 19 – Maxpooling d'une image 2D

Objectif: réduire la taille des caractéristiques extraites en sélectionnant la valeur maximale dans chaque région

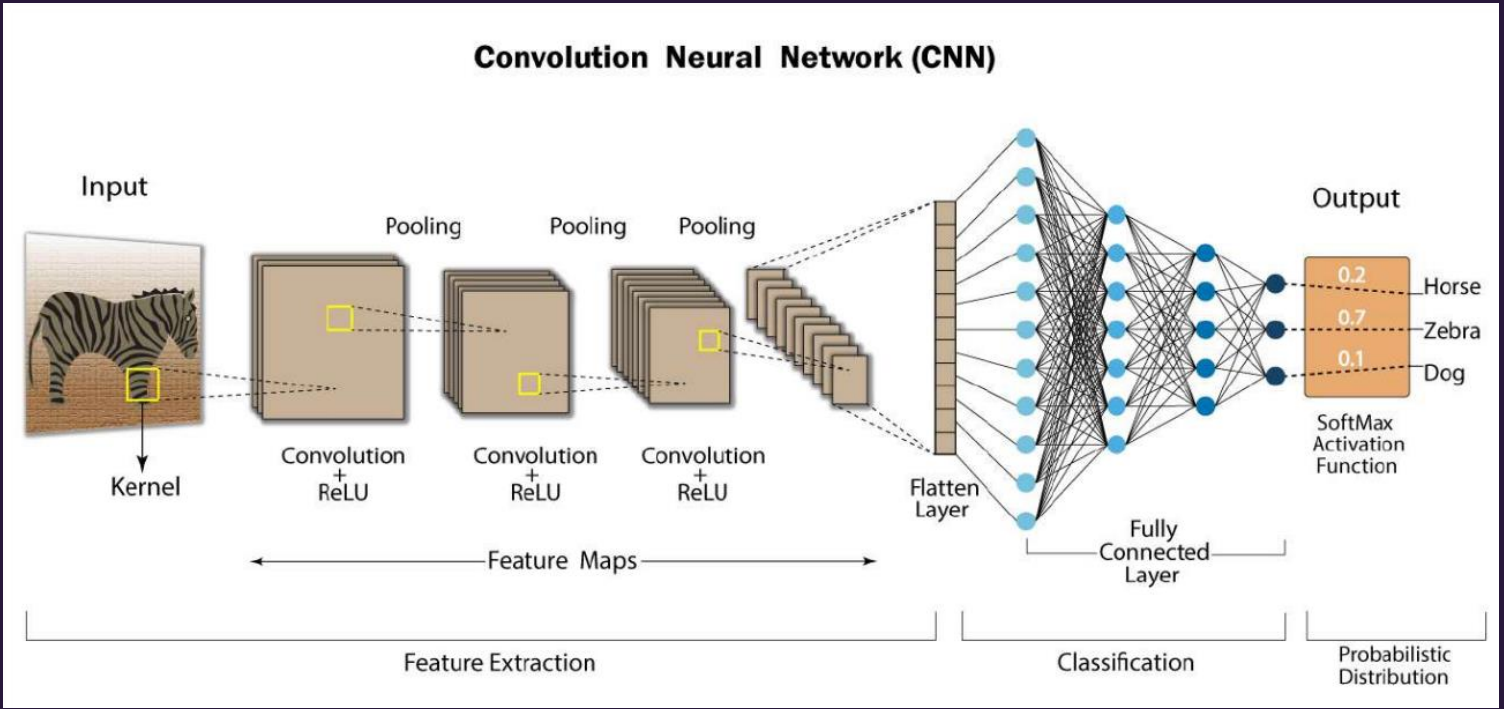


Figure 20 – Structure type d'un réseau de classification d'images

Expérience:

Collection des données:

Figure 21 – Balle «out», Rio Open 2020 (Lajovic vs Cecchinato) - terre battue

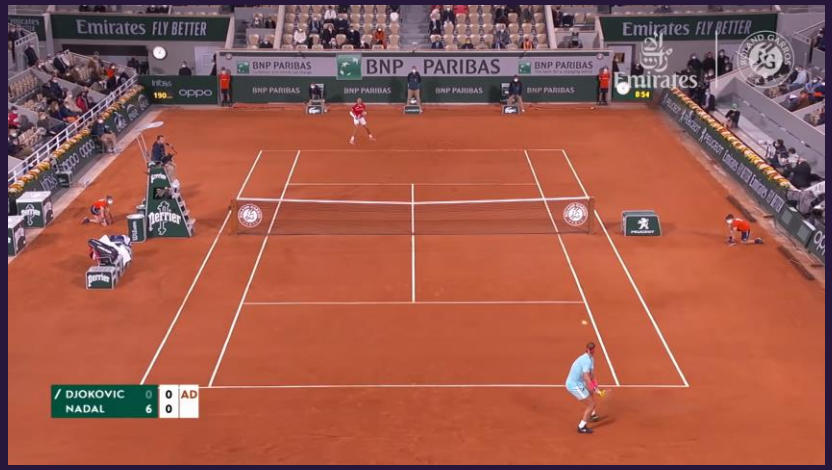


Figure 22 – Balle «in», (Nadal vs Djokovic) Final de Roland-Garros 2020 - terre battue

Expérience:

Figure 23 – Balle «in», terrain de tennis à proximité- terre battue

Figure 24 – Balle «out», terrain de tennis de notre lycée - terre battue

Nous avons pu collecter 612 images pour l'entraînement (terre battue) et 204 images pour le test (différentes surfaces)

Résultat:

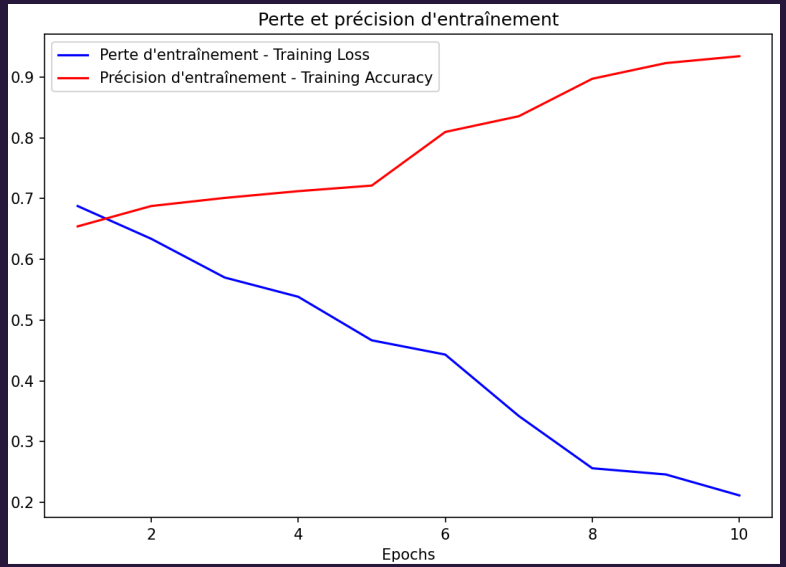


Figure 25 – Perte et précision d'entraînement

Nous avons atteint une précision de 96,07% à l'entraînement, cependant, la précision au test reste à améliorer avec 65,46 % .

```
Epoch 10/10
19/19 [==============================] - 38s 2s/step - loss: 0.1231 - accuracy: 0.9607
19/19 [==============================] - 11s 2s/step - loss: 0.5034 - accuracy: 0.6546
Test Accuracy: 65.46%
```

Sur-apprentissage:

Problème:

Se concentrer sur l'apprentissage à partir des données au détriment du but principal: **la prédiction**.

Exemple:

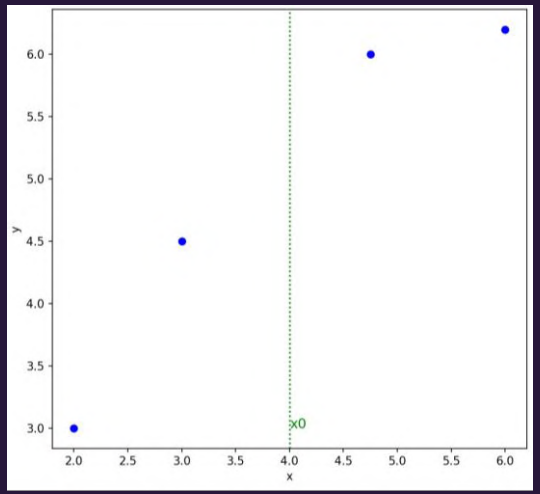


Figure 26 – Pour une valeur x0=4, on souhaite prédire une valeur y0 cohérente avec nos données.

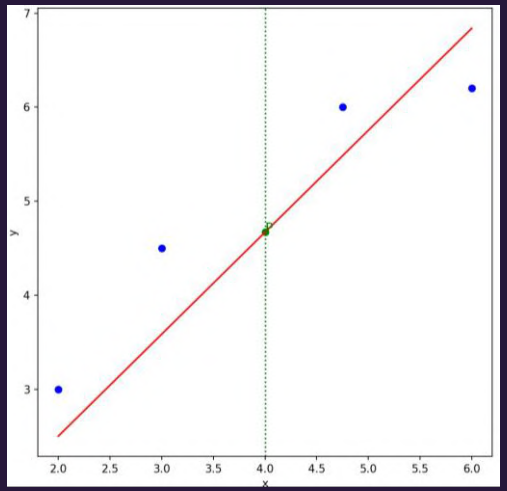


Figure 27 – Valeur prédite à l'aide d'une régression linéaire

Sur-apprentissage:

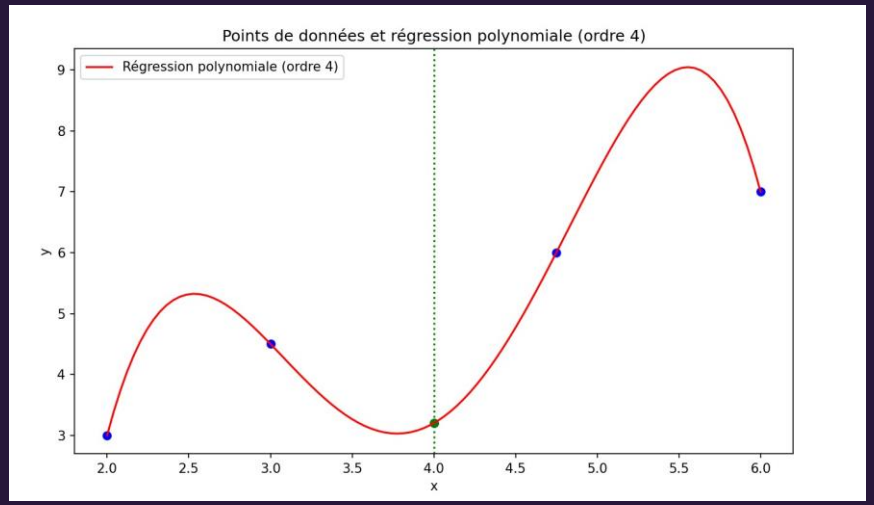


Figure 28 – Régression polynomiale (ordre 4) qui prédit une valeur y0 moins cohérente avec nos données.

Figure 29 – voiture roulant vite !

Solution proposée:

* Diversifier les données d'apprentissage:

Figure 30 – Balle «in», extraite d'un match d'entraînement entre Djokovic & Zverev (terrain dûr)

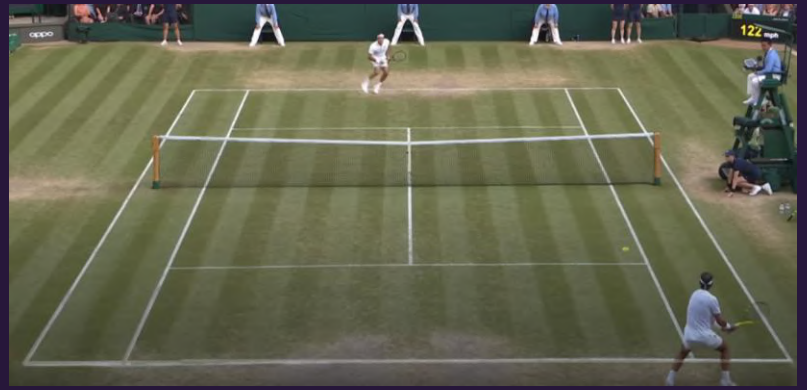


Figure 31 – Balle «in», dernier match entre Federer et Nadal à "Final Grand Slam Match" (terrain en gazon)

Solution proposée:

* Utilisation d'une méthode de régularisation: l'arrêt anticipé **(EarlyStopping)**

Principe:

- Diviser les données utilisées en 3 catégories:
  - Données d'entraînement **(70%)**
  - Données de validation **(15%)**
  - Données de test **(15%)**
- Arrêter l'entraînement d'un modèle lorsqu'il détecte que la **performance sur les données de validation** ne s'améliore plus.

Résultat:

```
Epoch 34/50
59/59 [==============================] - 118s 2s/step - loss: 0.3203 - accuracy: 0.8601 - val_loss: 0.4803 -
val_accuracy: 0.7980
10/10 [==============================] - 20s 2s/step - loss: 0.4852 - accuracy: 0.8109
Test Accuracy: 81.09%
```

Intégration pratique du modèle:

Figure 32 – Balle de tennis détectée à l'aide vision par ordinateur

Figure 33 – Balle de tennis frappant le sol (Cadre d'impact)

Changement de l'espace de couleurs

Principe: * Changement de représentation des images **(frames)** de **RGB (Rouge, Vert, Bleu)** à **HSV (Teinte, Saturation, Valeur)**

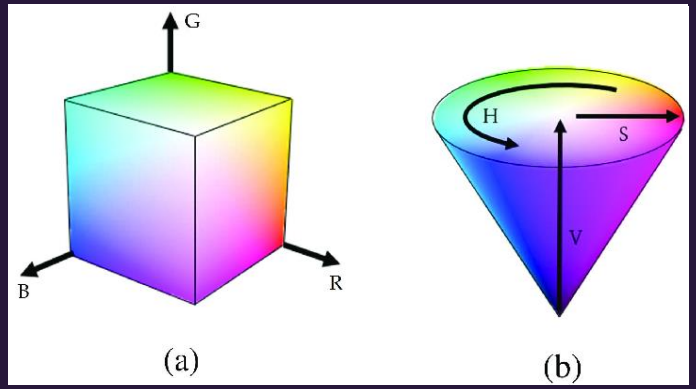


Figure 34 – Comparaison de l'espace de couleurs RGB en cube (a) et l'espace de couleurs HSV en cône (b)

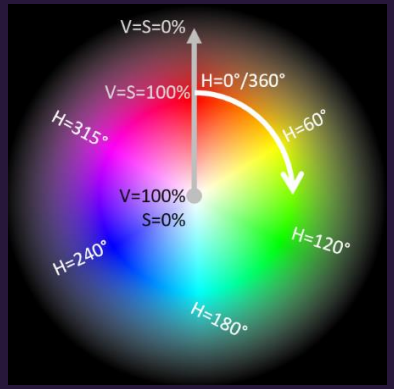


Figure 35 – Espace de couleurs HSV sous forme de cercle chromatique

Changement de l'espace de couleurs

Intérêt: - Séparation de la Couleur et de la Luminosité
=> Robustesse aux Variations de Luminosité **(la teinte reste relativement stable même si la luminosité de l'image change)**

Exemple:

Figure 36 – Représentation d'une image en RGB (Rouge, Vert, Bleu)

Figure 37 – Représentation d'une image en HSV (Teinte, Saturation, Valeur)

Seuil de Couleur (Color Thresholding):

Objectif: Isoler des objets spécifiques dans une image en fonction de leurs couleurs

Principe:

Définir une plage de valeurs de couleur pour créer **un masque binaire**

(en espace HSV: La détection est plus fiable sous différentes conditions d'éclairage)

$$\text{mask}(x,y) = \begin{cases} 1 & \text{si } (H_{lower} \le H(x,y) \le H_{upper}) \text{ et } (S_{lower} \le S(x,y) \le S_{upper}) \text{ et } (V_{lower} \le V(x,y) \le V_{upper}) \\ 0 & \text{sinon} \end{cases}$$

Seuil de Couleur (Color Thresholding):

Exemple:

Figure 38 – Le Français Ugo Humbert aux Demi-finales des Championnats de Tennis de Dubaï Duty Free 2024

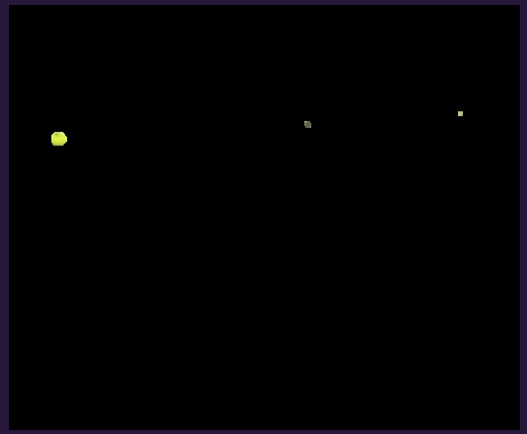


Figure 39 – Figure précédente après application du seuil

Figure 40 – Application d'une boîte englobante à la balle après sa détection

Application:

Objectif: Extraire le point d'impact d'une balle de tennis d'une vidéo à l'aide d'un traitement de vidéo

Principe:

* Appliquer **le masque binaire** sur les cadres de la vidéo
* Déterminer le cadre où la balle atteint **son minimum**

Computer vision

OpenCV

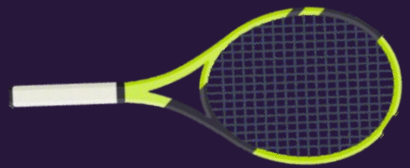
Résultat:

Conclusion:

Figure 41 – Erreur de la première approche

Figure 42 – Ball IN

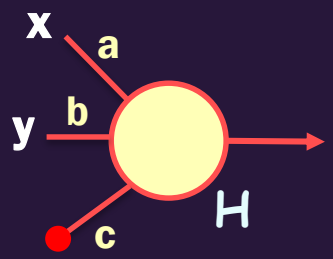
Merci pour votre attention



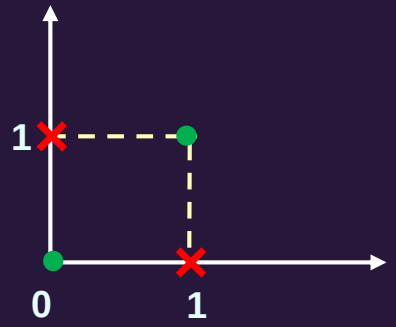
ANNEXE:

Un neurone pour «ou exclusif» ?

Supposons, par l'absurde, son existence:



$F(x,y)=H(ax+by+c)$



On a donc:

. $F(1,0) = H(a+c) = 1$	$\Rightarrow a+c \geq 0$	(1)
. $F(0,1) = H(b+c) = 1$	$\Rightarrow b+c \geq 0$	(2)
. $F(0,0) = H(c) = 0$	$\Rightarrow c < 0$	(3)
. $F(1,1) = H(a+b+c) = 0$	$\Rightarrow a+b+c < 0$	(4)

Mais, (1) + (2) + (3) => $a+b+c \geq 0$

(Absurde avec (4))

Représentation matricielle un réseau de neurones:

Soient:

$X=(x_1, ..., x_n)$: l'entrée d'une certaine couche

$Y=(y_1, ..., y_m)$: les valeurs prises à la couche suivante

On note W la matrice des poids des connections entre deux couches successives.

$$W=\begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} \text{ et } B=\begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

Ce qui nous donne:

$$\begin{cases} Y=\sigma(Wx+B) \\ y_j=\sigma(w_{i,j}x_i+b_i) \end{cases}$$

ANNEXE:

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

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def tanh(x):
    return np.tanh(x)
def relu(x):
    return np.maximum(0, x)

x = np.linspace(-10, 10, 400)
y_sigmoid, y_tanh, y_relu = sigmoid(x), tanh(x), relu(x)
plt.plot(x, y_sigmoid, label='Sigmoid Function')
plt.plot(x, y_tanh, label='Tanh Function')
plt.plot(x, y_relu, label='ReLU Function')
plt.title('Activation Functions')
plt.xlabel('x')
plt.ylabel('Function value')
plt.legend()
plt.grid(True)
plt.show()
```

Traçage des fonctions d'activation

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
x, y = np.meshgrid(x, y)
z = x**4 + y**2

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='viridis')
plt.show()
```

Traçage de la fonction g

ANNEXE:

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

def gradient_descent(f, grad_f, start, learning_rate=0.1, num_iterations=50):
    x, y = start
    path = [(x, y)]
    for _ in range(num_iterations):
        grad_x, grad_y = grad_f(x, y)
        x -= learning_rate * grad_x
        y -= learning_rate * grad_y
        path.append((x, y))
    return np.array(path)

def plot_gradient_descent(f, path, intermediate_steps=[1, 2, 50]):
    x = np.linspace(-3, 3, 400)
    y = np.linspace(-3, 3, 400)
    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)
    levels = np.logspace(0, 2, 20)

    fig, axes = plt.subplots(1, len(intermediate_steps), figsize=(18, 6))
    for i, step in enumerate(intermediate_steps):
        axes[i].contour(X, Y, Z, levels=levels)
        axes[i].plot(path[:step+1, 0], path[:step+1, 1], 'r.-')
        axes[i].set_title(f'Étape: {step}')
    plt.tight_layout()
    plt.show()

def f(x, y):
    return x**4 + y**2

def grad_f(x, y):
    return 4*x**3, 2*y

start = (2, 2)
path = gradient_descent(f, grad_f, start, learning_rate=0.1, num_iterations=50)
plot_gradient_descent(f, path, intermediate_steps=[0, 1, 50])
```

Application de la descente de gradient sur la fonction g

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def gradient_descent_3d(f, grad_f, start, learning_rate=0.1, num_iterations=50):
    x, y = start
    path = [(x, y)]
    for _ in range(num_iterations):
        grad_x, grad_y = grad_f(x, y)
        x -= learning_rate * grad_x
        y -= learning_rate * grad_y
        path.append((x, y))
    return np.array(path)

def plot_gradient_descent_and_surface(f, grad_f, path):
    x = np.linspace(-3, 3, 400)
    y = np.linspace(-3, 3, 400)
    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6)
    ax.plot(path[:, 0], path[:, 1], f(path[:, 0], path[:, 1]), 'r.-', markersize=5)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('f(X, Y)')
    plt.show()

def f(x, y):
    return x**4 + y**2

def grad_f(x, y):
    return 4*x**3, 2*y

start = (2, 2)
path = gradient_descent_3d(f, grad_f, start, learning_rate=0.1, num_iterations=50
plot_gradient_descent_and_surface(f, grad_f, path)
```

La représentation de cette dernière en 3D

ANNEXE:

```python
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt

# Chemin vers les données d'apprentissage et d'entraînement
train_data_dir = r'C:\Users\hp\Desktop\tipe test\ghmi9\train'
test_data_dir = r'C:\Users\hp\Desktop\tipe test\ghmi9\test'

img_width, img_height = 150, 150
batch_size = 32

# Augmentation des données pour l'entraînement
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Déterminons le pas par époque (steps per epoch)
steps_per_epoch = len(train_generator)
validation_steps = len(test_generator)
```

```python
# Définition du modèle
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Entraînement
history = model.fit(
    train_generator,
    epochs=10,
    verbose=1,
    steps_per_epoch=steps_per_epoch
)

# Évaluation du modèle
loss, accuracy = model.evaluate(
    test_generator,
    verbose=1,
    steps=validation_steps
)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Figure de perte et précision de l'entraînement
plt.figure(figsize=(14, 7))
plt.plot(history.history['loss'], label="Perte d'entraînement - Training Loss",
color='blue')
plt.plot(history.history['accuracy'], label="Précision d'entraînement - Training
Accuracy", color='red')
plt.title("Perte et précision d'entraînement")
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.tight_layout()
plt.show()
```


Le modèle utilisé pour l'entraînement (sans validation) / Evolution de la perte et précision d'entraînement

ANNEXE:

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X = np.array([2, 3, 4, 4.75, 6])
Y = np.array([3, 4.5, 3.2, 6, 7])
x0 = 4

X_reshaped = X.reshape(-1, 1)

plt.figure(figsize=(14, 7))

plt.subplot(1, 2, 1)
plt.scatter(X, Y, color='blue')
plt.axvline(x=x0, color='green', linestyle='dotted')
plt.text(x0, min(Y), 'x0', color='green', fontsize=12)
plt.title('Points de données et valeur x0')
plt.xlabel('x')
plt.ylabel('y')
```

```python
# modèle de régression linéaire ()
model = Sequential()
model.add(Dense(1, input_dim=1, activation='linear'))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_reshaped, Y, epochs=500, verbose=0)

# Prédictions
X_line = np.linspace(min(X), max(X), 100).reshape(-1, 1)
Y_line = model.predict(X_line)
y0 = model.predict(np.array([[x0]]))

# Deuxième schéma
plt.subplot(1, 2, 2)
plt.scatter(X, Y, color='blue')  # Points
plt.plot(X_line, Y_line, color='red')
plt.axvline(x=x0, color='green', linestyle='dotted')
plt.scatter([x0], y0, color='green')
plt.text(x0, y0, 'p', color='green', fontsize=12)
plt.title('Régression linéaire avec Keras')
plt.xlabel('x')
plt.ylabel('y')

plt.tight_layout()
plt.show()
```

Régression linéaire des 4 points et prédiction pour x0 (Sur-apprentissage)

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.polynomial.polynomial import Polynomial

X = np.array([2, 3, 4, 4.75, 6])
Y = np.array([3, 4.5, 3.2, 6, 7])
x0 = 4

coeffs = Polynomial.fit(X, Y, 4).convert().coef

x_fit = np.linspace(min(X), max(X), 100)
y_fit = np.polyval(coeffs[::-1], x_fit)

plt.figure(figsize=(14, 7))
plt.scatter(X, Y, color='blue')
plt.scatter(x0, np.polyval(coeffs[::-1], x0),
color='green')
plt.axvline(x=x0, color='green', linestyle='dotted')
plt.plot(x_fit, y_fit, color='red', label='Régression
polynomiale (ordre 4)')
plt.title('Points de données et régression polynomiale
(ordre 4)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Régression polynomiale (même objectif)

ANNEXE:

```python
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Chemins vers les répertoires de données
train_data_dir = r'C:\Users\hp\Desktop\tipe test\ghmi9\train'
validation_data_dir = r'C:\Users\hp\Desktop\tipe test\ghmi9\validation'
test_data_dir = r'C:\Users\hp\Desktop\tipe test\ghmi9\test'

# Dimensions des images et taille des lots
img_width, img_height = 150, 150
batch_size = 32

# Générateur de données avec augmentation pour l'entraînement
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Générateur de données pour la validation et le test (seulement rescaling)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)
```

```python
# Calcul du nombre de pas par epoch
steps_per_epoch = train_generator.samples // batch_size
validation_steps = validation_generator.samples // batch_size

# Définition du modèle
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Ajout d'un callback EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Entraînement du modèle avec validation et EarlyStopping
history = model.fit(
    train_generator,
    epochs=5,
    validation_data=validation_generator,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    verbose=1,
    callbacks=[early_stopping]
)

# Évaluation du modèle sur les données de test
test_loss, test_accuracy = model.evaluate(test_genrator, verbose=1)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

Le modèle utilisé pour l'entraînement (avec validation et arrêt anticipé)

ANNEXE:

```python
import cv2
import os

def rgb_to_hsv_image(input_image_path, output_folder_path):
    image = cv2.imread(input_image_path)
    if image is None:
        print("Error: Unable to read the image.")
        return
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    base_name = os.path.basename(input_image_path)
    output_image_path = os.path.join(output_folder_path, f"hsv_{base_name}")
    cv2.imwrite(output_image_path, hsv_image)

input_image_path = r"C:\Users\hp\Desktop\tipe plots\7mida-rass-l7lwa.jpg"
output_folder_path = r"C:\Users\hp\Desktop\tipe plots"

rgb_to_hsv_image(input_image_path, output_folder_path)
```

Transformation d'une image RGB ne HSV

```python
import cv2
import numpy as np
import os

def process_image(image_path, output_folder):
    image = cv2.imread(image_path)
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # plage de couleurs pour la balle de tennis jaune
    H_lower, H_upper = 25, 35
    S_lower, S_upper = 50, 255
    V_lower, V_upper = 50, 255

    mask = cv2.inRange(hsv, (H_lower, S_lower, V_lower), (H_upper, S_upper, V_upper)) # masque binaire

    # réduction des impuretés par des opérations morphologiques
    kernel = np.ones((5, 5), np.uint8)
    mask_cleaned = cv2.erode(mask, kernel, iterations=1)
    mask_cleaned = cv2.dilate(mask_cleaned, kernel, iterations=1)

    result_cleaned = cv2.bitwise_and(image, image, mask=mask_cleaned) # Application du masque nettoyé à l'image originale

    os.makedirs(output_folder, exist_ok=True)
    output_mask_cleaned_path = os.path.join(output_folder, 'mask_cleaned_output.jpg')
    output_result_cleaned_path = os.path.join(output_folder, 'result_cleaned_output.jpg')

    cv2.imwrite(output_mask_cleaned_path, mask_cleaned)
    cv2.imwrite(output_result_cleaned_path, result_cleaned)

    return output_mask_cleaned_path, output_result_cleaned_path

image_path = r'C:\Users\hp\Desktop\tipe plots\Dubai-2024-Tennis-ATP-Day-5-Ugo-Humbert.jpg'
output_folder = r'C:\Users\hp\Desktop\tipe plots'

mask_path, result_path = process_image(image_path, output_folder)
```

Isolation de la balle par le seuil de couleur (Ugo Humbert)

ANNEXE:

```python
import cv2
import numpy as np
import os

def detect_ball_using_color(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # RGB en HSV

    lower_yellow = np.array([25, 50, 50])
    upper_yellow = np.array([35, 255, 255])         # plage du jaune

    mask = cv2.inRange(hsv, lower_yellow, upper_yellow)     # masque binaire

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)    # contours dans le masque
    ball_position = None
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > 50:  # Filtrage des petites zones
            x, y, w, h = cv2.boundingRect(contour)
            ball_position = (x, y, w, h)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    return frame, ball_position

def track_ball_and_find_lowest_point(input_video_path, output_video_path, output_frame_path):
    cap = cv2.VideoCapture(input_video_path)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_video_path, fourcc, 30.0, (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))))

    lowest_y = float('-inf')
    lowest_frame = None
```

```python
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        tracked_frame, ball_position = detect_ball_using_color(frame) # Détection de la balle à l'aide de la couleur

        if ball_position:
            _, y, _, _ = ball_position
            if y > lowest_y:  # la coordonnée y la plus élevée pour le point le plus bas
                lowest_y = y
                lowest_frame = frame.copy()

        out.write(tracked_frame) # le cadre avec la balle détectée dans la vidéo de sortie

    cap.release()
    out.release()

    if lowest_frame is not None:
        cv2.imwrite(output_frame_path, lowest_frame) # Enregistrement du cadre le plus bas
        return output_frame_path

    return None

input_video_path = r'C:\Users\hp\Desktop\vidtennis1.mp4'
output_folder = r'C:\Users\hp\Desktop\tipe_last_dance'
output_video_path = os.path.join(output_folder, 'vidéo_suivie.mp4')
output_frame_path = os.path.join(output_folder, 'cadre_point_le plus bas.jpg')

os.makedirs(output_folder, exist_ok=True)
extracted_frame_path = track_ball_and_find_lowest_point(input_video_path, output_video_path, output_frame_path)
print("Le cadre extrait est sauvegardé à :", extracted_frame_path) # Résultat d'affichage
```

Extraction du cadre où la balle touche le sol