ACADÉMIE
DE PARIS

Liberté
Égalité
Fraternité

Faisons académie !

Journée du Numérique

L'IA en pratique

VM-FL

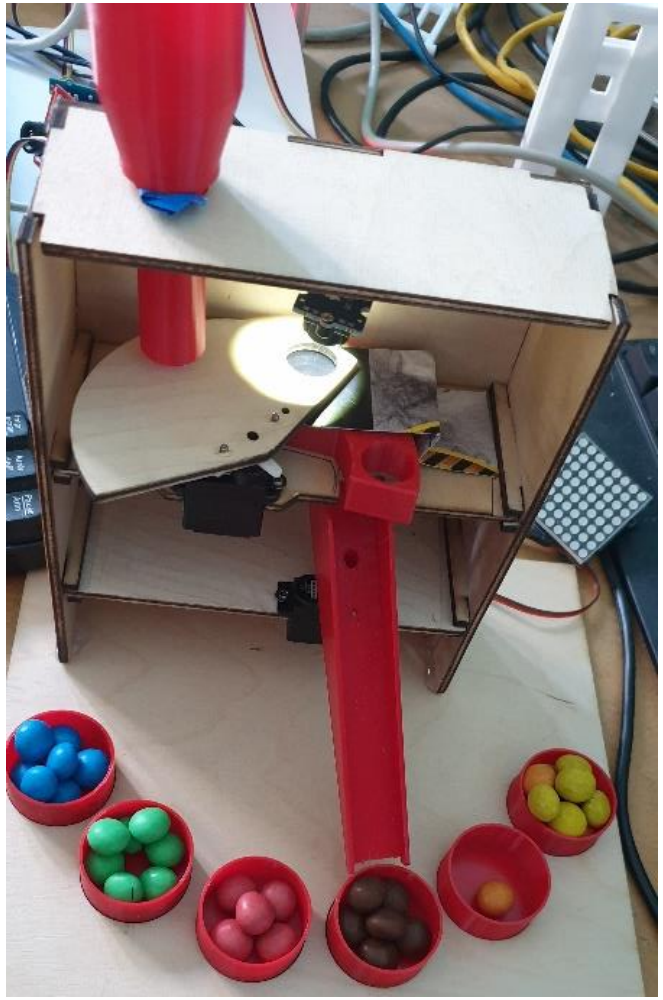
Utilisation de l'IA dans des applications pratiques

SCAI sur le campus de Jussieu

Mercredi 12 février 2025

ACADÉMIE
DE PARIS
Liberté
Égalité
Fraternité

SOMMAIRE



- **Objectifs de la présentation :**

  Pourquoi l'IA en classe de SSI ?
- **Présentation du projet de trieur de M&Ms**

  Montrer l'intérêt de l'IA dans des projets pédagogiques pratiques
- **Utilisation de l'IA pour améliorer le système**

  Approche avec un modèle d'IA classique
- **Introduction au Deep Learning**

  Approche avec un réseau de neurones convolutif (CNN) :
- **Intégration et applications pédagogiques**

  Intégration du système complet :
- **Conclusion et échanges**

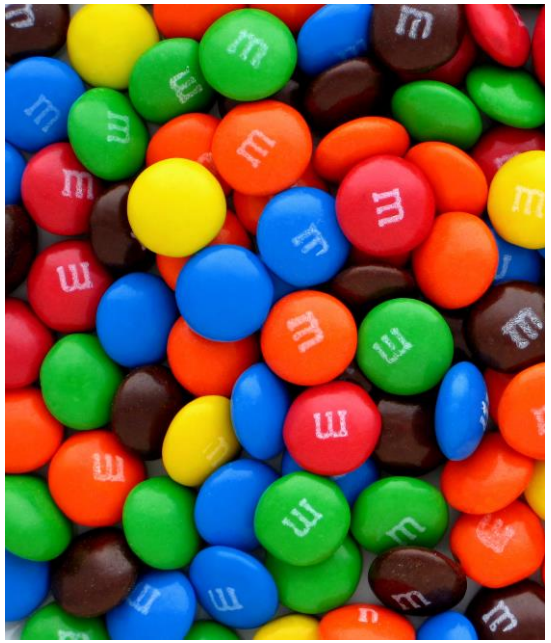
ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Objectifs de la présentation :

•Montrer l'intérêt de l'IA dans des projets pédagogiques pratiques.

•Comparer une solution classique avec des approches utilisant l'IA et le deep learning.

•Illustrer l'application concrète avec un trieur de couleurs pour des M&M's.



Pourquoi l'IA en classe de SSI ?

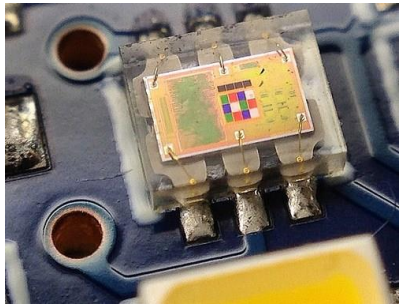
•Relever les compétences pour le futur (industrie, recherche, quotidien).

•Développement de compétences transversales (mathématiques, algorithmique, analyse).

•Possibilité de rendre les projets plus attractifs et engageants.

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

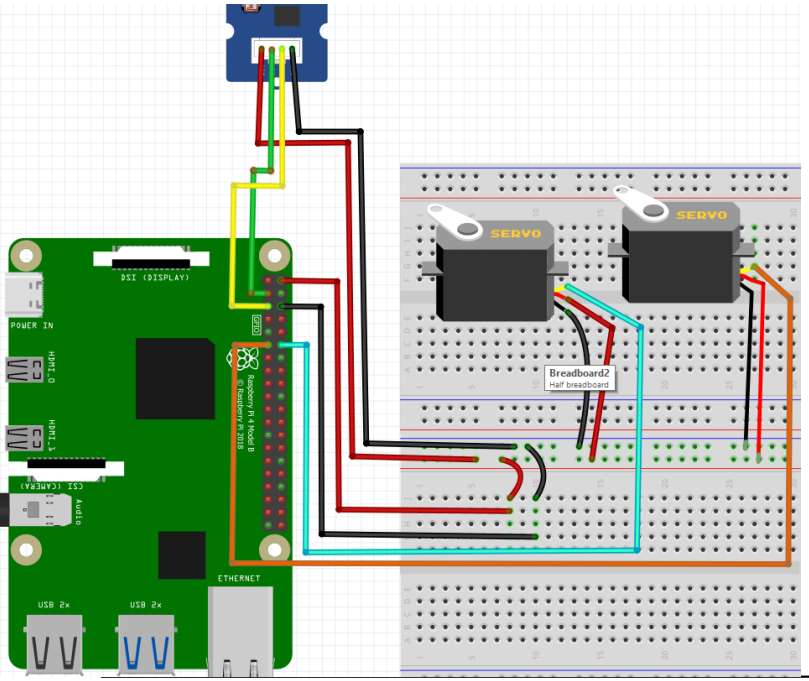
Description du système :



•Fonctionnement d’un trieur de couleurs pour des M&M's.

•Utilisation d’un Raspberry Pi comme plateforme de développement.

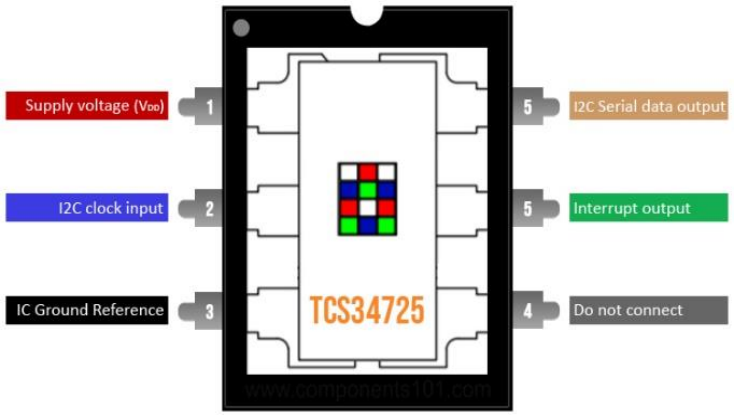
•Acquisition des données : capteur RGB.

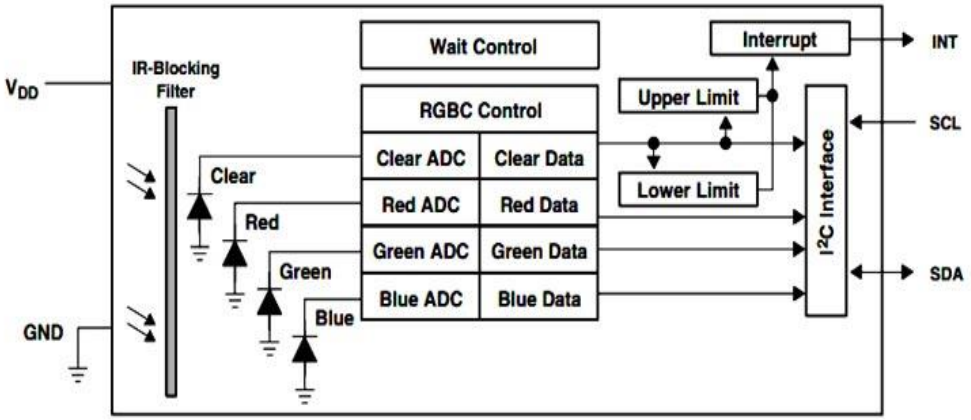


Solution classique :

- Mesure des valeurs de couleur (RVB ou HSL).
- Comparaison par seuils ou calcul d’écart entre les valeurs mesurées et des valeurs de référence pour chaque couleur.
- Limites de cette approche :
- Sensibilité aux variations de lumière.
- Complexité si les couleurs se chevauchent dans l’espace RVB.

Détection de lumière réfléchie :





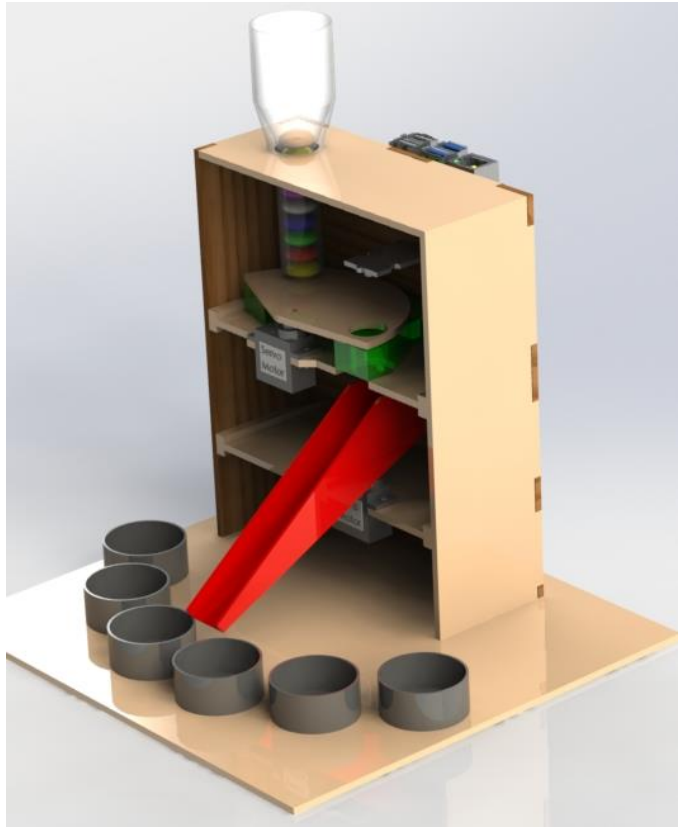
- **Éclairage intégré :**

  Le TCS34725 possède une LED blanche intégrée pour fournir un éclairage constant, améliorant la précision des mesures indépendamment des conditions lumineuses ambiantes.

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Approche classique : Analyse des écarts de valeurs mesurées

Maquette de détection des couleurs



Création d'un fichier csv avec lecture des couleurs

1. **Servo 1** : avec trois positions :
   - Position 1 : Chargement.
   - Position 2 : Mesure.
   - Position 3 : Éjection.
2. **Servo 2** :
   - 6 positions pour trier les M&M's par couleur : marron, vert, orange, jaune, rouge, bleu.
3. **Capteur** TCS34725

```
Red,Green,Blue,Couleur
0.0,2.0,5.0,bleu
0.0,2.0,5.0,bleu
-1.0,0.0,4.0,bleu
0.0,0.0,4.0,bleu
0.0,0.0,4.0,bleu
0.0,0.0,4.0,bleu
0.0,1.0,4.0,bleu
0.0,1.0,5.0,bleu
0.0,1.0,5.0,bleu
0.0,0.0,4.0,bleu
6.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
5.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
6.0,-1.0,0.0,rouge
6.0,0.0,1.0,rouge
6.0,0.0,0.0,rouge
```

Principe de création du fichier csv

1. **Initialisation :**
   - Le capteur et les servos sont initialisés dans les positions par défaut.
2. **Collecte des Échantillons :**
   - Pour chaque couleur dans la liste `COULEUR`, le programme lit les données RGB pour 10 pastilles (ou une valeur ajustable dans `num_samples`).
   - Entre chaque lecture, le servo horizontal (`servoh`) est déplacé pour la position de mesure de déchargement et de chargement.
3. **Enregistrement des Données :**
   - Une fois toutes les pastilles mesurées, les données sont enregistrées dans un fichier CSV avec des colonnes pour la couleur et les valeurs RGB.

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Détermination des écarts de couleurs :

Programme

```
import csv
# Fonction pour calculer les plages de variation des couleurs
def calculate_color_ranges(file_path):
    color_data = {}
    # Lecture du fichier CSV
    with open(file_path, 'r', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        next(reader)  # Ignorer les en-têtes
        for row in reader:
            r, g, b, color = float(row[0]), float(row[1]), float(row[2]), row[3]
            if color not in color_data:
                color_data[color] = {"R": [], "G": [], "B": []}
            color_data[color]["R"].append(r)
            color_data[color]["G"].append(g)
            color_data[color]["B"].append(b)
    # Calcul des plages pour chaque couleur
    color_ranges = {}
    for color, channels in color_data.items():
        color_ranges[color] = {
            "R": (min(channels["R"]), max(channels["R"])),
            "G": (min(channels["G"]), max(channels["G"])),
            "B": (min(channels["B"]), max(channels["B"]))
        }
    return color_ranges
# Fonction principale
def main():
    file_path = "couleurs_echantillons.csv"  # Remplacez par le chemin de
votre fichier CSV
    color_ranges = calculate_color_ranges(file_path)

    print("Plages de variation des couleurs :")
    for color, ranges in color_ranges.items():
        print(f"{color.capitalize()} :")
        print(f"  R (Rouge) : {ranges['R']}")
        print(f"  G (Vert)  : {ranges['G']}")
        print(f"  B (Bleu)  : {ranges['B']}")
# Exécution du script
if __name__ == "__main__":
    main()
```

Fonctionnement

- **Lecture des données CSV :**

Le fichier CSV est lu ligne par ligne, et les valeurs RGB associées à chaque couleur sont stockées dans un dictionnaire.

- **Calcul des plages de variation :**

Pour chaque couleur, les valeurs minimales et maximales de `R`, `G` et `B` sont déterminées.

- **Affichage des résultats :**

Les plages de variation pour chaque couleur sont imprimées de manière lisible.

Ecart min et max pour les 6 couleurs

	R		G		B	
	min	max	min	max	min	max
Color						
bleu	0.0	0.0	1.0	2.0	4.0	5.0
jaune	10.0	12.0	10.0	13.0	2.0	3.0
marron	0.0	1.0	-1.0	-1.0	0.0	1.0
orange	8.0	10.0	3.0	4.0	1.0	1.0
rouge	6.0	7.0	0.0	1.0	0.0	1.0
vert	1.0	2.0	4.0	5.0	2.0	2.0

Utilisation dans le programme pour les 6 couleurs

```
COLOR_RANGES = {
    "bleu": [(0.0, 0.0), (1.0, 2.0), (4.0, 5.0)],
    "rouge": [(6.0, 7.0), (0.0, 1.0), (1.0, 1.0)],
    "vert": [(2.0, 2.0), (5.0, 6.0), (2.0, 2.0)],
    "marron": [(0.0, 1.0), (-1.0, -1.0), (0.0, 1.0)],
    "jaune": [(10.0, 12.0), (10.0, 13.0), (2.0, 3.0)],
    "orange": [(9.0, 11.0), (4.0, 5.0), (1.0, 2.0)],
}
```

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Tri par fourchette de couleurs:

Fonctionnement

1. **Servo 1** :
   - Change entre 3 positions (chargement, mesure, éjection) pour gérer le flux des M&M's.
   - Les angles pour chaque position sont définis dans `max mid et min`.

2. **Détection des couleurs** :
   - Les couleurs sont détectées en comparant les valeurs de rouge, vert et bleu.
   - Des seuils sont définis pour certaines couleurs complexes comme jaune, orange ou marron.

3. **Servo 2** :
   - Positionné selon la couleur détectée : marron, vert, orange, jaune, rouge, ou bleu.
   - Les angles pour chaque couleur sont définis dans `SERVO2_POSITIONS`.

1. **Tri** :
   - Le premier servo gère le déplacement des M&M's entre les étapes.
   - Le second servo place les M&M's dans le compartiment correspondant.

Programme

Gestion de la position et des écarts

```
servob_POSITIONS = {
    "marron": -0.3,   # Position pour marron
    "vert": 0.6,      # Position pour vert
    "orange": -0.95,  # Position pour orange
    "jaune": -0.7,    # Position pour jaune
    "rouge": 0.1,     # Position pour rouge
    "bleu": 0.9       # Position pour bleu
}

COLOR_RANGES = {
    "bleu": [(0.0, 0.0), (1.0, 2.0), (4.0, 5.0)],
    "rouge": [(6.0, 7.0), (0.0, 1.0), (1.0, 1.0)],
    "vert": [(2.0, 2.0), (5.0, 6.0), (2.0, 2.0)],
    "marron": [(0.0, 1.0), (-1.0, -1.0), (0.0, 1.0)],
    "jaune": [(10.0, 12.0), (10.0, 13.0), (2.0, 3.0)],
    "orange": [(9.0, 11.0), (4.0, 5.0), (1.0, 2.0)],

}
```

Gestion du tri avec une tolérance de 1

```
# Déterminer la couleur dominante
def detect_color():
  _, r, g, b = read_colors()
  print(f"R: {r}, G: {g}, B: {b}")
  tolerance = 1
  for color, (r_range, g_range, b_range) in COLOR_RANGES.items():
        if (
            (r_range[0] - tolerance) <= r <= (r_range[1] + tolerance) and
            (g_range[0] - tolerance) <= g <= (g_range[1] + tolerance) and
            (b_range[0] - tolerance) <= b <= (b_range[1] + tolerance)
        ):
            return color
  return "inconnu"
```

TRI OK avec une tolérance de 1 mais des couleurs sont mesurées plusieurs fois

Tolérance de 2 pas de pb de lectures mais des couleurs sont mal reconnues

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Introduction au modèles d'IA classiques

Machine Learning

- **Principe :**
  Remplacer la logique conditionnelle par un modèle d'IA entraîné pour classer les couleurs.

- **Modèles abordés :**
  - Régression logistique. LR
  - Forêt aléatoire.
  - k-Nearest Neighbors (k-NN).
  - LDA
  - Naïve Bayes
  - SVM

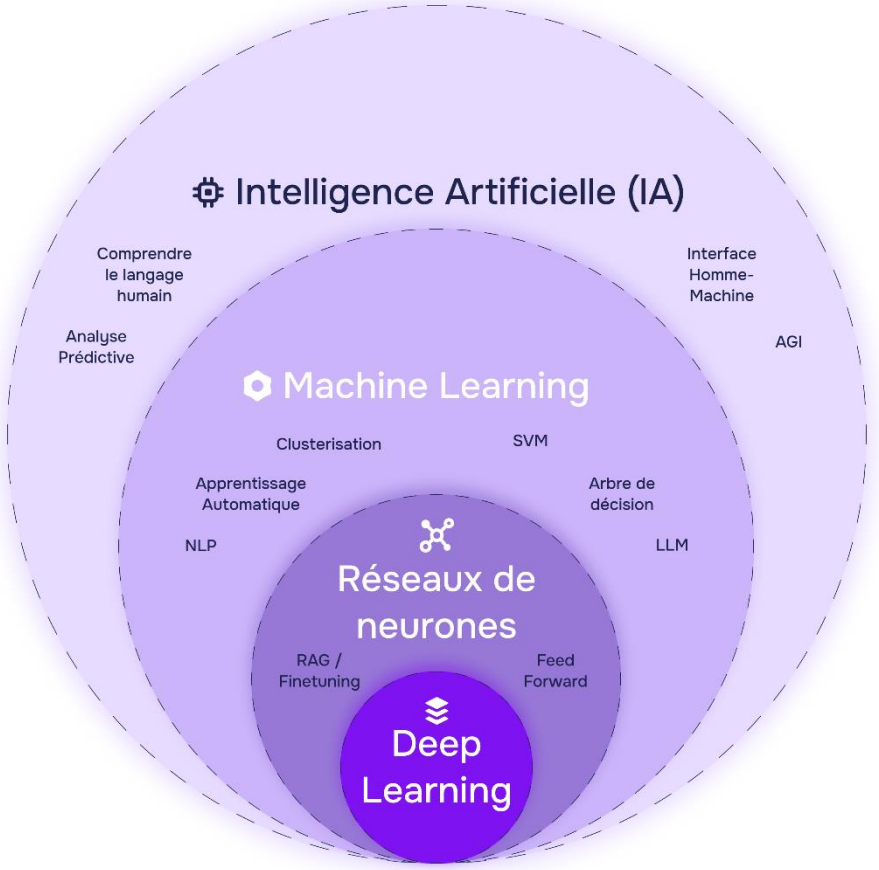
Quatre **étapes à réaliser :**

1. **Collecte de données :** Créer un dataset en mesurant des valeurs RVB et en annotant manuellement les couleurs.
2. **Entraînement des modèles :** Utiliser une bibliothèque comme `scikit-learn` pour entraîner un modèle sur les données collectées.
3. **Évaluation :** Comparer les performances des différents modèles sur des M&M's non vus pendant l'entraînement.
4. **Implémentation :** Intégrer le modèle choisi dans le programme pour trier les M&M's en temps réel.

Résumé : Quand utiliser quel modèle ?

Modèle	Complexité	Données linéaires	Données non linéaires	Interprétabilité	Rapidité
Régression logistique	Faible	Oui	Non	Élevée	Rapide
LDA	Moyenne	Oui	Non	Moyenne	Moyenne
k-NN	Faible	Oui	Oui	Moyenne	Lent
Arbre de décision	Moyenne	Oui	Oui	Élevée	Moyenne
Naïve Bayes	Faible	Oui	Non	Moyenne	Rapide
SVM	Élevée	Oui	Oui (avec noyaux)	Faible	Lent

Schéma de fonctionnement d'une IA



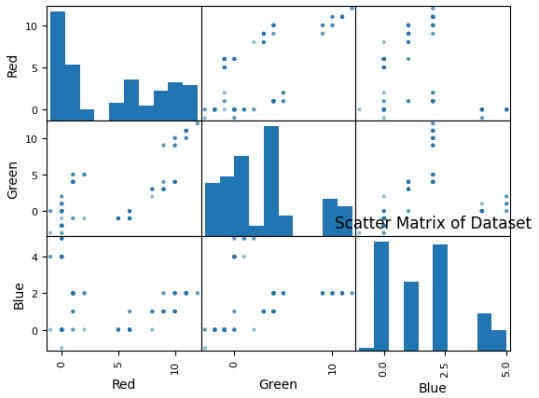
2024 - ©DataBird - www.data-bird.co

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Entrainement du modèle

Fonctionnement

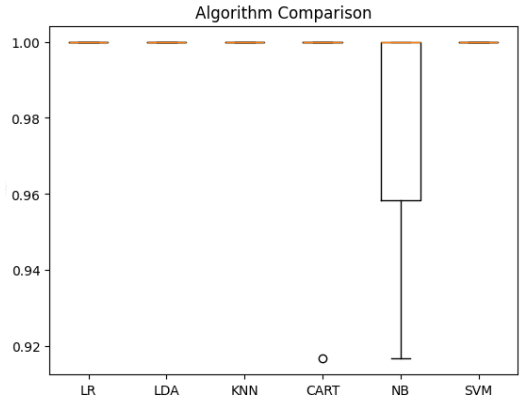
1. **Chargement des données** : Ajout de gestion d'erreur pour s'assurer que le fichier CSV existe avant de continuer.
2. **Exploration des données** : Ajout d'une matrice de dispersion pour visualiser les relations entre les caractéristiques.
3. **Préparation des données** : Utilisation explicite de `iloc` pour séparer les caractéristiques et les labels.
4. **Comparaison des modèles** : Résultats affichés avec moyenne et écart type. Les graphiques de comparaison sont ajoutés
   Entraînement et sauvegarde : Fonction utilitaire pour éviter la duplication de code, avec des évaluations détaillées pour chaque modèle.



Utilisation des bibliothèques Sklearn

```
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split,
cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import LabelEncoder
import joblib
```

```
# Matrice de dispersion des variables
scatter_matrix(dataset)
plt.title("Scatter Matrix of Dataset")
plt.show()
```




```
# === 2. Préparation des données ===

# Séparation des caractéristiques (RVB) et des labels
(classe des couleurs)
X = dataset.iloc[:, :3].values  # Colonnes RVB
Y = dataset.iloc[:, 3].values  # Classe de couleur

# Encodage des labels (classes des couleurs)
label_encoder = LabelEncoder()
Y_encoded = label_encoder.fit_transform(Y)

# Séparation des données en ensembles d'entraînement et
de validation
X_train, X_validation, Y_train, Y_validation =
train_test_split(
    X, Y_encoded, test_size=0.30, random_state=1
)

# === 3. Définition et évaluation des modèles ===

# Liste des modèles à tester
models = [
    ('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')),
    ('LDA', LinearDiscriminantAnalysis()),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier()),
    ('NB', GaussianNB()),
    ('SVM', SVC(gamma='auto'))
]
```

Exploitation des résultats

Régression logistique. LR

```
Exploitationmodele = joblib.load('LR.pkl')
label_encoder = joblib.load('label_encoder_LR.pkl')
```

Confusion rouge et orange un marron dans les verts

Analyse Discriminante Linéaire (**LDA**)

```
modele = joblib.load('DA.pkl')
label_encoder = joblib.load('label_encoder_LDA.pkl')
```

Tri OK pas d'erreur

k-Nearest Neighbors (**KNN**) :

```
modele = joblib.load('KNN.pkl')
label_encoder = joblib.load('label_encoder_KNN.pkl')
```

Tri OK pas d'erreur

Naïve Bayes (**NB**) :

```
modele = joblib.load('NB.pkl')
label_encoder = joblib.load('label_encoder_NB.pkl')
```

Confusion vert, marron et bleu un vert dans les oranges

Arbre de Décision (**CART**) :

```
modele = joblib.load('KNN.pkl')
label_encoder = joblib.load('label_encoder_KNN.pkl')
```

Confusion marron et rouge, un vert dans les oranges

Support Vector Machines (**SVM**)

```
modele = joblib.load('SVM.pkl')
label_encoder = joblib.load('label_encoder_SVM.pkl')
```

Tri OK pas d'erreur

Introduction aux réseaux de neurones

Un réseau de neurones est un modèle d'intelligence artificielle inspiré du fonctionnement du cerveau humain, conçu pour résoudre des problèmes complexes en apprenant des données.

1.Structure :

- Il est composé de **couches de neurones** :
  - Une couche d'entrée (reçoit les données brutes).
  - Une ou plusieurs couches cachées (effectuent des transformations intermédiaires).
  - Une couche de sortie (fournit le résultat ou la prédiction).
- Chaque neurone dans une couche est connecté à ceux de la couche suivante.

2.Traitement des données :

- Chaque connexion entre neurones est pondérée par un **poids**, qui détermine l'importance d'une entrée spécifique.
- Chaque neurone applique une **fonction d'activation** pour introduire des non-linéarités et permettre au réseau de modéliser des relations complexes.

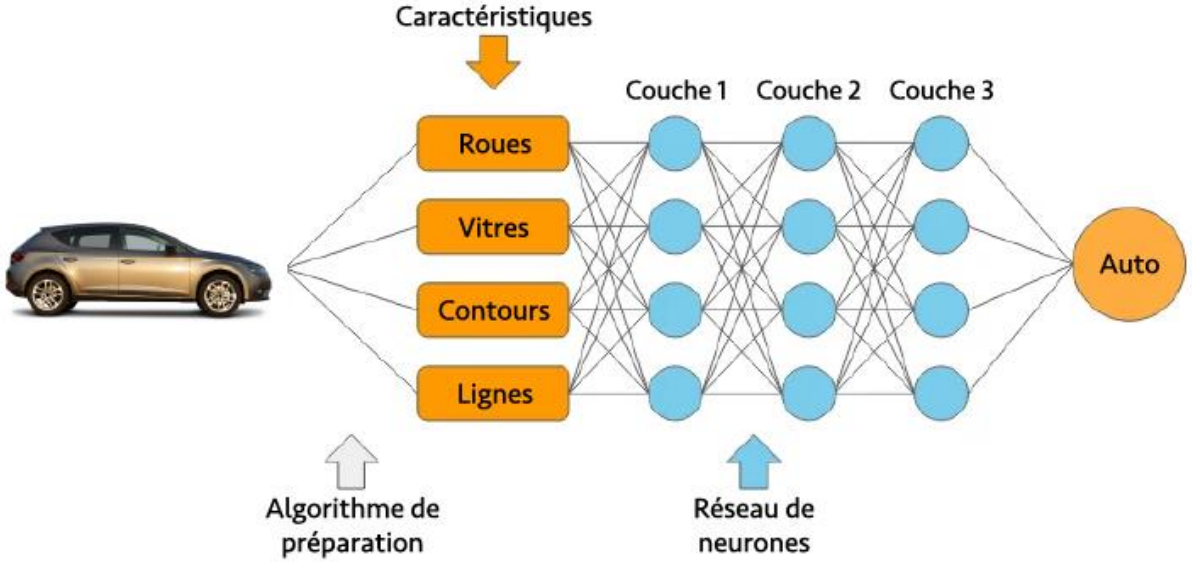
3.Apprentissage :

- Lors de l'entraînement, le réseau ajuste les poids en fonction de l'erreur entre sa prédiction et le résultat attendu.
- Un **algorithme d'optimisation** (comme la rétro propagation et la descente de gradient) met à jour les poids pour réduire cette erreur.

4.Objectif :

- À mesure que le réseau apprend, il devient capable de généraliser les relations dans les données pour faire des prédictions sur de nouvelles données.

En résumé, un réseau de neurones est un modèle d'apprentissage automatique qui affine ses connexions internes pour identifier des patterns et fournir des prédictions ou classifications à partir de données d'entrée.



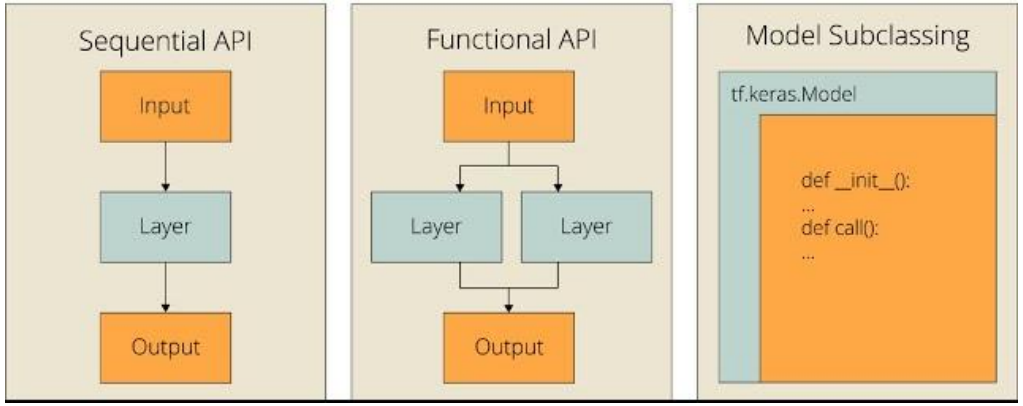

```
import tensorflow as tf

# Modèle simple
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Compilation
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Entraînement
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val, y_val))
```

ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Utilisation avec tensorflow

Principe : Utiliser un réseau de neurones pour classer les couleurs avec davantage de flexibilité.
Etapes principale :

1. Préparation des données
2. Création du model avec Tensoflow/Keras
3. Entrainement du modèle
4. Evaluation du modèle
5. Utilisation du model pour prédire des couleurs

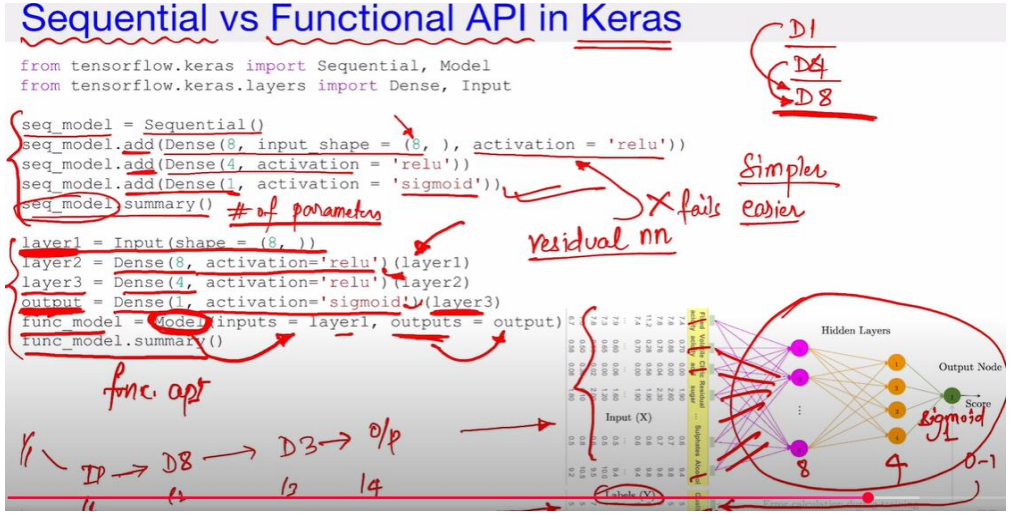


Model séquentiel

```
# Définir le modèle TensorFlow

model = Sequential([
    Dense(64, input_dim=3, activation='relu'),  # Couche d'entrée
    Dropout(0.2),  # Dropout pour éviter le surapprentissage
    Dense(32, activation='relu'),  # Couche cachée
    Dense(y_one_hot.shape[1], activation='softmax')  # Couche de sortie
])

# Compiler le modèle
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
```

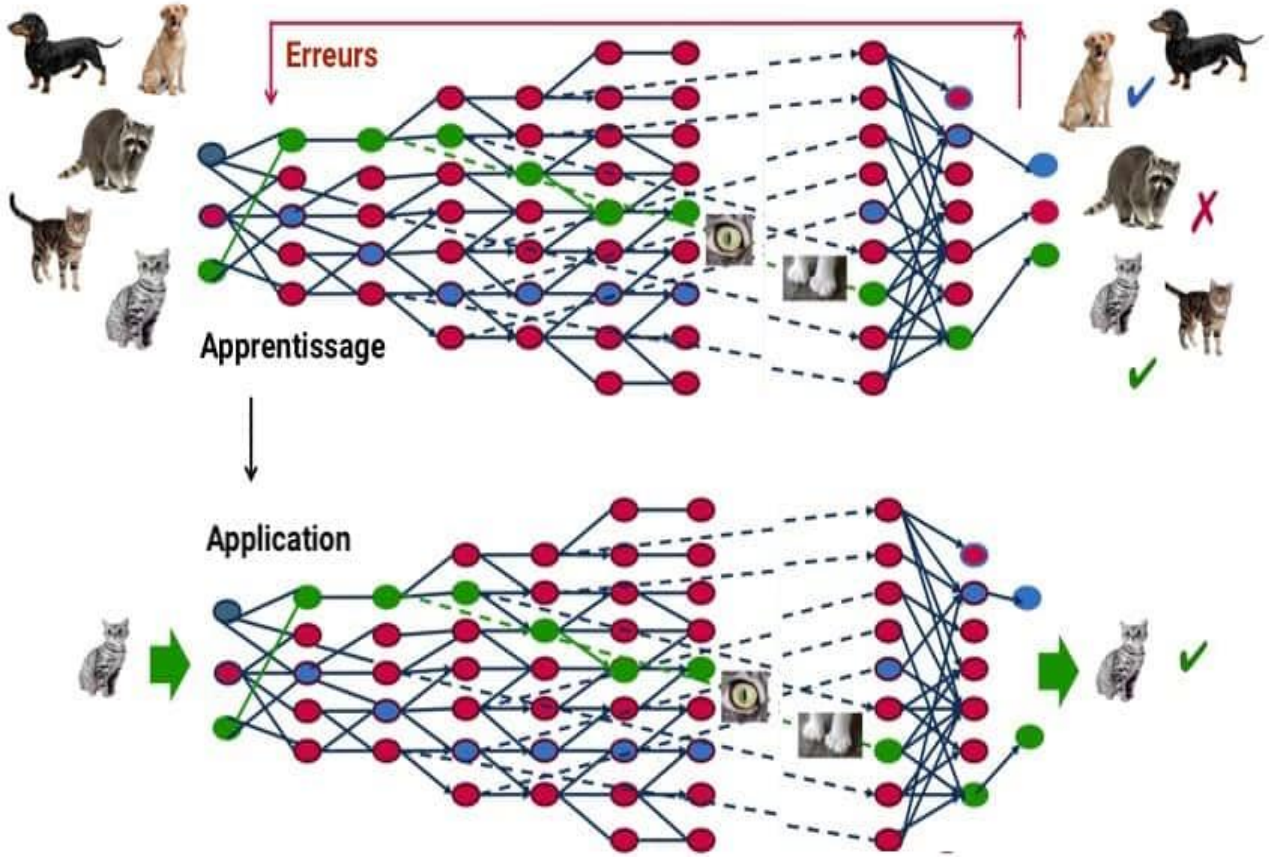
Model API

```
# Définir le modèle en utilisant l'API fonctionnelle
inputs = Input(shape=(3,), name="Input_Layer")  # Couche d'entrée
x = Dense(64, activation='relu', name="Dense_1")(inputs)  # Première couche dense
x = Dropout(0.2, name="Dropout_1")(x)  # Dropout pour régularisation
x = Dense(32, activation='relu', name="Dense_2")(x)  # Deuxième couche dense
outputs = Dense(y_one_hot.shape[1], activation='softmax', name="Output_Layer")(x)  # Couche de sortie

# Création du modèle
model = Model(inputs=inputs, outputs=outputs)
```



Introduction au Deep learning

Le **deep learning** est une sous-catégorie de l'intelligence artificielle qui repose sur des réseaux de neurones. Ces réseaux sont particulièrement adaptés pour apprendre des représentations complexes à partir des données brutes, sans nécessiter de règles explicites ou d'étiquetage manuel.



Dans le cas d'un trieur de couleurs :

- **Données d'entrée** : Les valeurs brutes mesurées par un capteur RGB (ou les pixels d'une image capturée).
- **Traitement** :
  - Le réseau de neurones apprend à identifier des motifs dans les données (par exemple, des regroupements naturels des couleurs).
  - Les couches internes du réseau découvrent des représentations implicites des caractéristiques de chaque couleur.
- **Décision** : Le réseau décide à quelle catégorie (couleur) appartient l'échantillon, sans qu'il soit nécessaire de fournir des règles explicites.

Deep Learning avec tensor flow

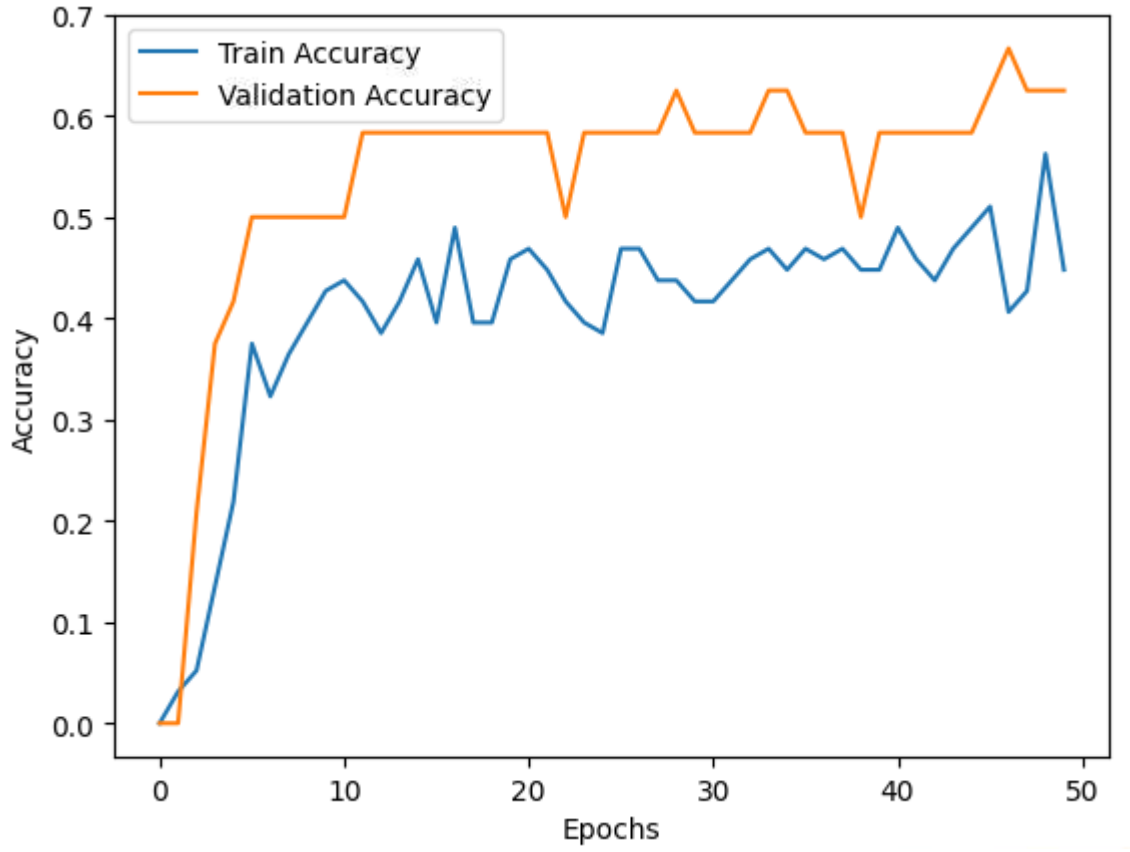
Chargement et Préparation des Données

```
# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, random_state=42)

# Définir le modèle TensorFlow
model = Sequential([
    Dense(64, input_dim=3, activation='relu'),  # Couche d'entrée avec 3 neurones pour RGB
    Dropout(0.2),  # Dropout pour éviter le surapprentissage
    Dense(32, activation='relu'),  # Couche cachée
    Dense(y_one_hot.shape[1], activation='softmax')  # Couche de sortie
])

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Indices des classes dans les données d'entraînement : [0, 2, 4, 5, 6, 8, 10, 11, 12, 15, 17, 24, 25, 27, 28, 29]

```
"vertical": {
    25: -0.94,  # Jaune
    27: -0.7,   # Orange
    15: -0.3,   # Marron
    9: 0.1,     # Rouge
    28: 0.1,    # Rouge
    12: 0.55,   # Vert
    13: 0.55,   # Vert
    4: 0.95     # Bleu
}
```

```
Epoch 50/50
12/12 [==============================] - 0s 31ms/step - loss: 1.3584 - accuracy: 0.48
oss: 1.8538 - val_accuracy: 0.5833
Indices des classes dans les données d'entraînement : [0, 2, 4, 5, 6, 8, 10, 11, 12,
, 25, 27, 28, 29]

Évaluation sur l'ensemble de test :
1/1 [==============================] - 1s 1s/step - loss: 1.8538 - accuracy: 0.5833
Précision : 58.33%
```



ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

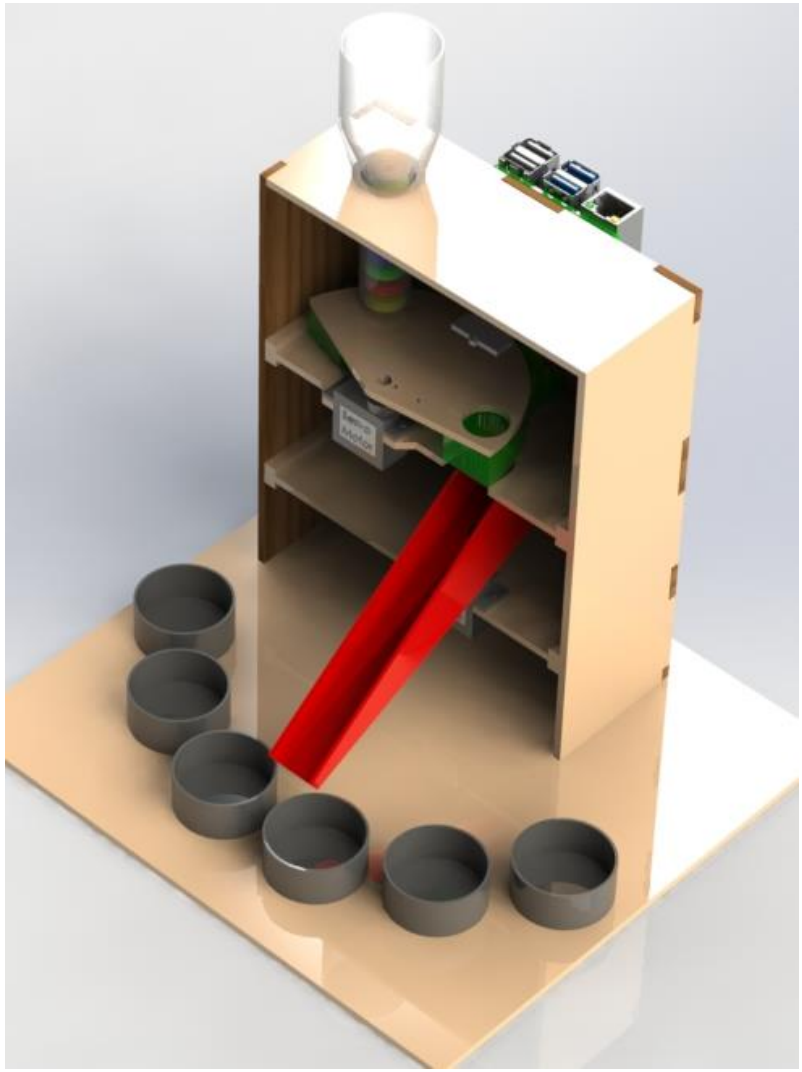
Deep Learning avec tensor flow

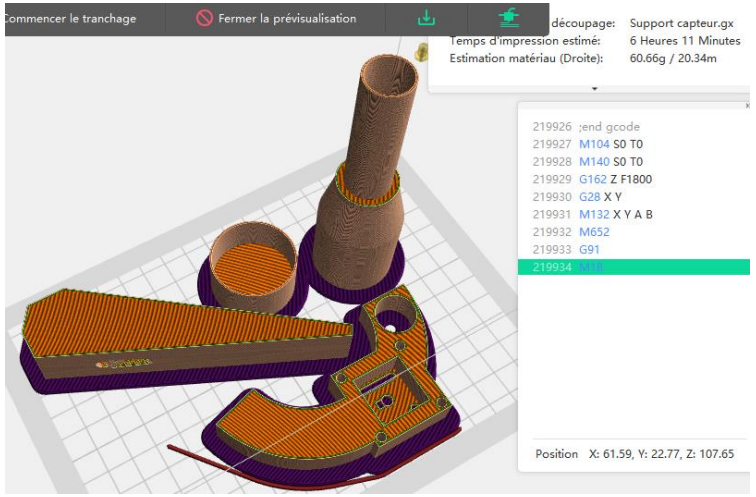
```
# Fonction : Détecter l'index de la couleur
def detect_color_index():
    red, green, blue = read_colors()
    # Préparer les données pour le modèle TensorFlow
    nouvelle_couleur = [[red, green, blue]]  # Forme attendue : liste de
listes

    # Faire la prédiction
    prediction = modele.predict(nouvelle_couleur)
    print(f"Probabilités de la prédiction : {prediction[0]}")
    couleur_index = tf.argmax(prediction[0]).numpy()  # Index de la
classe prédite
    print(f"Index de la couleur détectée : {couleur_index}")
    return couleur_index

# Fonction : Positionner les servomoteurs
def position_servo(servo, position):
    servo.value = position
    sleep(0.5)
```

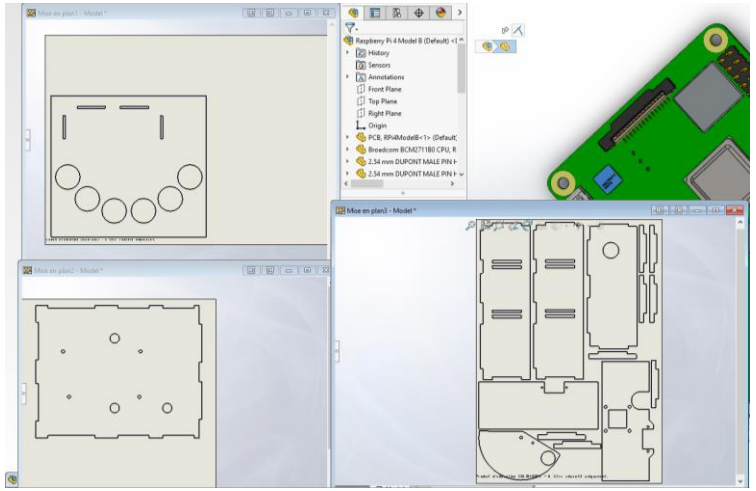
```
# Fonction : Trier les M&M's
def tri_m_and_ms():
    while True:
        print("Déplacement en position de chargement...")
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["chargement"])

        print("Déplacement en position de mesure...")
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["mesure"])
        couleur_index = detect_color_index()

        if couleur_index in SERVO_POSITIONS["vertical"]:
            print(f"Tri pour la couleur d'index : {couleur_index}")
            position_servo(servo_vertical, SERVO_POSITIONS["vertical"][couleur_index])
            print("Éjection...")
            position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["ejection"])
        else:
            print(f"Couleur inconnue (index {couleur_index}), éjection à une position par défaut.")
            position_servo(servo_vertical, 0)  # Position par défaut ou d'échec

        sleep(1)  # Pause avant la prochaine itération
```

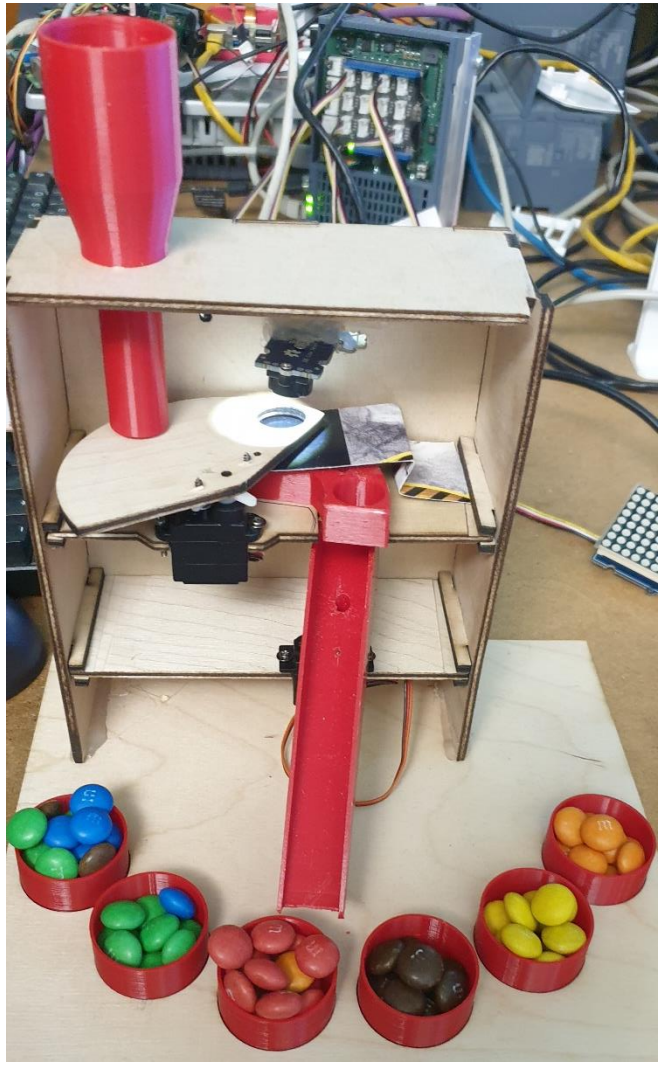
ACADÉMIE DE PARIS
Liberté
Égalité
Fraternité

Application pédagogique









ACADÉMIE
DE PARIS
Liberté
Égalité
Fraternité

MERCI DE VOTRE ATTENTION