Smart Environments - AutoMiam

Cahier des charges

Encadrant : Lionel Médini

Étudiants : Titouan Knockaert et Gaspard Goupy

M2 Intelligence artificielle - Université Claude Bernard Lyon 1

AutoMiam

Table des matières

Contexte

Ce projet est réalisé dans le cadre de l'enseignement *Smart Environments*, enseigné au sein du Master 2 Intelligence Artificielle de l'université Claude Bernard Lyon 1. Il est encadré par Lionel Medini et a pour sujet les objets connectés, l'internet des objets (*IoT*), et le web des objets (*WoT*).

Les distributeurs de nourriture pour animaux, aussi appelés *pet feeders*, ne sont pas des technologies récentes. En effet, leur première commercialisation date du XXe siècle, comme illustre cette publicité pour le *Kenl-Mastr automatic pet feeder*, datant de 1939.

Cependant, les *pet feeders* ont connu un essor récent, notamment grâce à la démocratisation des objets intelligents et connectés du quotidien. Ils sont désormais commercialisés par centaines, offrant des fonctionnalités diverses, telles que la régulation de nourriture, la minuterie ou encore le contrôle à distance. Néanmoins, leurs capacités sont bien souvent assez simples, et peu intègrent des fonctionnalités complexes, réellement intelligentes.

Ainsi, il serait intéressant d'intégrer des techniques d'IA actuelles dans ces systèmes, afin d'obtenir des comportements intelligents. Dans ce projet, nous allons chercher à complexifier un *pet feeder* à l'aide de méthodes de *Deep Learning*.

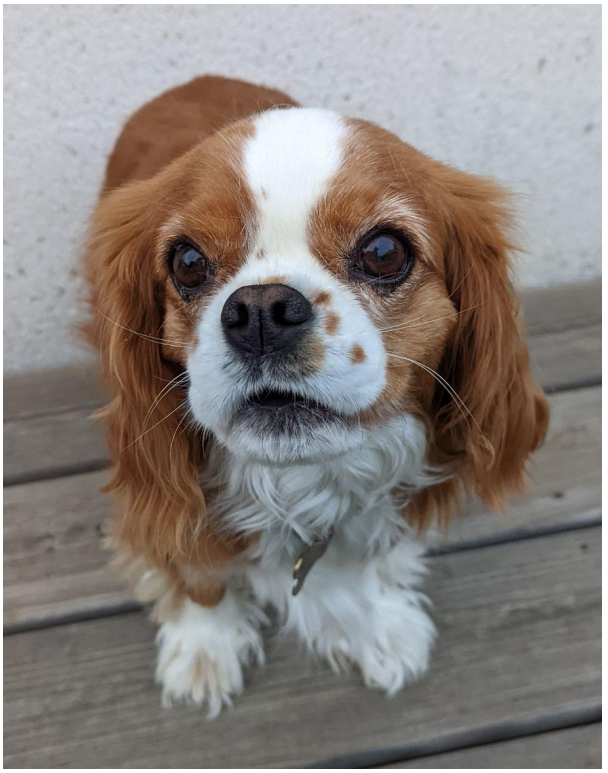
Résumé

L'objectif de ce projet est de réaliser un *pet feeder* intelligent, capable d'identifier les animaux enregistrés, afin d'adapter son comportement pour chacun.

Le système pourra délivrer de la nourriture, de manière automatique, aux animaux enregistrés qui s'en approche. Avant de nourrir un animal, il l'identifiera à l'aide d'une caméra pour analyser ses besoins et lui fournir une dose de nourriture quotidienne adaptée. Le maître pourra enregistrer ses animaux dans le système et suivre leur consommation grâce à une interface web.

Dans le cadre du projet, nous restreindrons les animaux aux chiens, et le nombre de chiens reconnus à deux : *Mina* et *Jappeloup*.

Jappeloup



Mina

Disclaimer : par ce projet, nous n'encourageons pas l'utilisation des distributeurs de nourriture pour animaux. Aucune utilisation personnelle du prototype ne sera faite. Par ailleurs, aucun animal n'a été maltraité durant la réalisation du projet.

Scénario

A - Scénario classique, centré sur le chien :

1. Le *pet feeder* est en attente.

2. Une entité -*supposée chien*- s'approche du *pet feeder*, jusqu'à arriver dans la distance d'activation, mesurée par un changement de luminosité.

3. Le *pet feeder* prend une photo de l'entité et procède à son identification avec des modèles de *Deep Learning*.

4.1 - Aucun chien n'est identifié, le *pet feeder* se remet en attente (1).

4.2 - Un chien est identifié, le *pet feeder* analyse sa consommation quotidienne et donne une dose de nourriture si l'animal n'a pas atteint son quota.

B - Scénario classique centré, sur le maître :

1. Le maître souhaite connaître la consommation actuelle de ses chiens.

2. Il se rend sur l'interface web utilisateur.

3. Dans l'onglet "*home*", il a accès à la consommation actuelle et maximale par jour, pour chacun de ses chiens.

Extensions :

4. Dans l'onglet "*pet-feeder*", il a accès aux informations relatives au système, notamment à la caméra, au nombre de doses données sur la journée, ou encore l'heure à laquelle la dernière dose à été donnée.

5. Dans l'onglet "*settings*", il peut enregistrer un nouveau chien dans le système, en indiquant son nom, sa quantité maximale de nourriture quotidienne, ainsi qu'une photo de l'animal pour permettre son identification.

Configuration Matérielle

- **Carte arduino modèle "*Uno*"** : gestion des capteurs et actionneurs
- **Servomoteur** : actionneur permettant, lors de son activation, de délivrer une dose de nourriture
- **Photorésistance** : capteur pour la détection de variation de luminosité
- **Caméra** : capteur pour l'identification des chiens
- **Ordinateur** : hébergement des applications et modules

Technologies logicielles

1. *Module "Pet Feeder"* : module en charge de l'objet connecté.

A - Thing : application exposant l'objet connecté selon la spécification *WoT* du W3C. L'API permet de lire les valeurs des capteurs et de contrôler les actionneurs de l'objet.
Langage : *Javascript*, *Typescript*
Technologies : *Node.js*, *Thingweb*, *Johnny-Five*

B - Controller : application consommant l'objet connecté, selon la spécification WoT du W3C. Contient la logique du *pet feeder* et le contrôle.
Langage : *Javascript*
Technologies : *Node.js*, *Thingweb*

2. *Module "Dog Identifier"* : application pour l'identification des chiens. Expose une API permettant d'enregistrer, d'identifier et de récupérer les données sur un chien. → ***Service "web" externe utilisé***
Langage : *Python*
Technologies : *flask*, *Tensorflow*, *Keras*

3. *Interface web utilisateur* : pages web servant de feedback pour l'utilisateur.
Langage : *Html*, *css*, *Javascript*

Standards WoT du W3C

Nous utiliserons des standards *WoT* du *W3C* pour concevoir le module *Pet Feeder*. Ainsi, en adoptant un formalisme standardisé, l'objet connecté pourrait être utilisé par des applications externes.

- *Thing Description* : définit un modèle formel pour représenter l'interface (propriétés et capacités) de l'objet connecté.

- *Scripting API* & *Binding templates* : pour exposer et consommer l'objet connecté, à l'aide de *Servients*.

Nous utiliserons Thingweb, une implémentation *Node.js* du *WoT*. L'objet sera exposé au format *JSON-LD*.

Intelligence ajoutée

La caractéristique principale de notre *pet feeder* est sa capacité à identifier les chiens enregistrés, afin d'adapter son comportement pour chacun.

Le premier comportement ajouté est la prise de décision sur la distribution de nourriture. En effet, chaque chien enregistré est limité quotidiennement par un seuil à ne pas dépasser, afin de ne pas dépenser son salaire en croquettes. Parallèlement, le système est désactivé (*freezé*) pendant un temps déterminé après une activation, pour étaler la distribution sur la journée. La *figure 1.* illustre ce comportement.

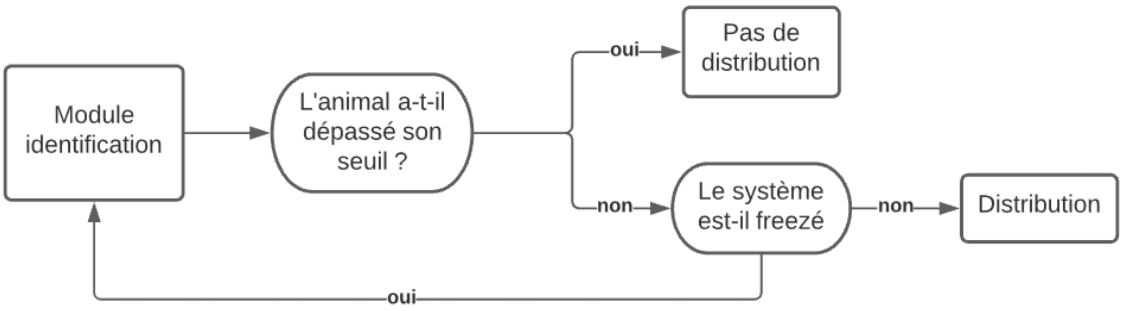


figure 1. décision sur la distribution de nourriture

Le deuxième et principal comportement ajouté est la capacité de détection, reconnaissance et identification des chiens enregistrés.

- La détection d'une entité (supposée chien) à proximité du *pet feeder* est réalisée à l'aide d'un capteur de luminosité (*Photorésistance*) : une variation de luminosité indique (potentiellement) la présence d'un chien.

- La reconnaissance d'un chien est réalisée par un réseau de neurones. Son objectif est de classifier une image donnée, pour reconnaître, ou non, une race de chien. Le modèle utilisé est *ResNet*50V2, fourni par la librairie *Keras* et pré-entraîné avec la base de données *imagenet*. Il peut détecter plus de 1000 classes, dont 117 races de chiens.

- L'identification d'un chien enregistré est réalisée par un réseau de neurones *Siamois*. Son objectif est de comparer une image donnée avec les images des chiens enregistrés, pour trouver lequel est le plus similaire (avec un palier minimum). Les sous réseaux utilisés sont des *ResNet*50V2 de *Keras*. L'architecture d'un réseau Siamois est représentée *figure* 2.

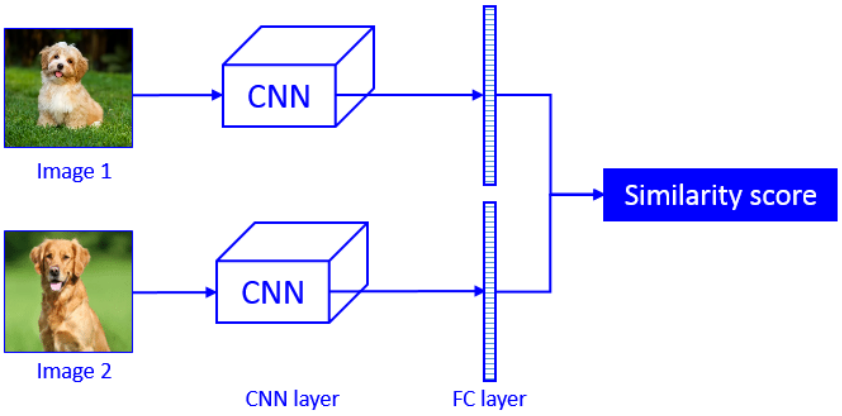


figure 2. Architecture d'un réseau de neurones Siamois

La *figure* 3. illustre le processus d'identification d'un chien.

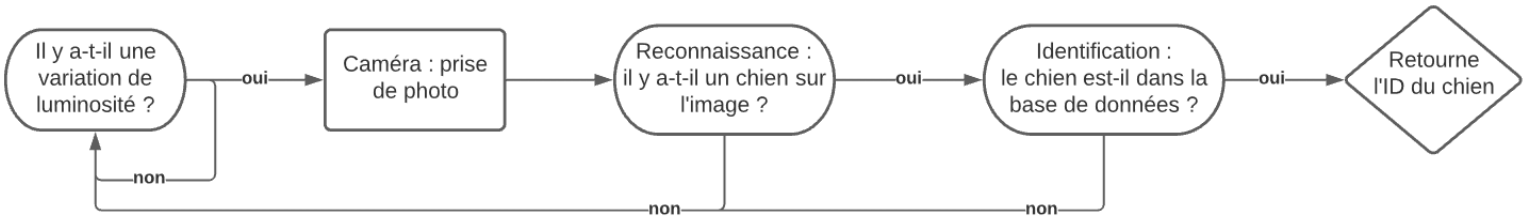


figure 3. processus d'identification d'un chien

Bilan

Pour conclure sur ce projet, nous pouvons discuter de plusieurs points d'implémentation.

Dans un premier temps, le système réalisé possède un certain nombre d'avantages. Tout d'abord, il est modulable, chaque application ayant un objectif précis et unique (*eg.* exposer l'objet connecté, identifier et gérer les chiens enregistrés, etc.). Ainsi, il est très facile d'ajouter une couche ou de réutiliser un module pour une autre application. Plus précisément, le fait que l'objet connecté soit exposé selon les spécifications *WoT* du *W3C* offre la possibilité d'être utilisé par des applications externes, ce qui s'inscrit notamment dans le cadre d'un développement d'objets connectés pour une *Smart House*. On pourrait alors imaginer une intégration Google Assistant. Par ailleurs, le système d'identification des chiens est bien étudié. L'utilisation d'un réseau de neurones *Siamois*, apprenant à mesurer une similarité plutôt qu'à classifier, permet une adaptation globale, quand il est bien entraîné (nous reviendrons plus tard sur ce point). Ainsi, il pourrait être utilisé par d'autres personnes sans avoir besoin de ré-entraîner le modèle. Par exemple, dans le cadre d'une commercialisation qui n'arrivera jamais. Finalement, le système obtient de très bonnes performances sur l'identification d'un chien. Le taux de précision mesuré est de 98% pour l'identification deux chiens différents et de 90% pour deux chiens similaires. L'objectif principal est d'obtenir la précision la meilleure possible sur les différences, afin de minimiser les erreurs du système sur la distribution au mauvais chien.

Dans un second temps, nous pouvons analyser les inconvénients du système réalisé. Tout d'abord, l'utilisation d'une photorésistance n'est pas adaptée à la détection de présence. Nous l'avons utilisé car nous n'avions pas de capteur de mouvement à notre disposition. Malheureusement, la photorésistance dépend de la luminosité ambiante et les variations ne sont pas toujours bien détectées, en particulier dans les endroits sombres. Par ailleurs, la caméra utilisée est de mauvaise qualité et ne capture pas assez de lumière, empêchant l'identification dans les espaces peu lumineux. À noter qu'utiliser la caméra de l'ordinateur est aussi un inconvénient, mais nous n'avions pas de caméra pour arduino. Par ailleurs, le fond (*background*) des images capturées, bien que réduit par le pré-processing du modèle, peut influencer l'identification. Idéalement, il faudrait traiter l'image pour récupérer uniquement la tête du chien. Cela peut être fait avec un réseau de neurones, ou par des méthodes d'analyse d'images classiques. En pratique, il faudrait entraîner notre modèle sur des têtes de chiens uniquement, et orienter la caméra de façon à ne voir sa tête que s'il est suffisamment proche du système, afin de ne pas activer la distribution dès qu'un chien passe devant, sans réellement vouloir manger.

Finalement, le projet peut faire l'œuvre de plusieurs améliorations. Nous pourrions dans un premier temps nous intéresser aux inconvénients cités précédemment : un meilleur capteur de présence, une meilleure caméra, un pré-processing précis des images, etc. De plus, nous pourrions utiliser une caméra à brancher sur arduino, afin d'améliorer la modularité. En effet, actuellement, notre caméra se trouve sur l'ordinateur et non sur l'objet connecté, ce qui empêche d'utiliser l'un sans l'autre. Par ailleurs, nous avons évoqué dans la première section que le réseau de neurones *Siamois* permet une adaptation globale quand il est bien entraîné. Dans notre cas, nous n'avons pu l'entraîner qu'avec des photos de Mina et Jappeloup. En réalité, il faudrait utiliser une grande base de données de chiens, contenant une trentaine de photos pour chacun, afin de faire apprendre au réseau à mesurer une différence sur deux images de chiens de manière générale. Nous pourrions même élargir le concept aux animaux domestiques en général, tels que les chats, auquel cas il faudrait aussi les intégrer à la base de données. Il est donc important de pré-process les images pour les centrer sur la tête de l'animal, afin d'obtenir de meilleures chances de détecter les similarités et les différences de chacun. Pour finir, nous pourrions aussi travailler le module UI : le développement web n'est pas vraiment notre spécialité…