isep
École d'ingénieurs du numérique

Station automatisée de tri des déchets

CORREC Romeo & TABLEAU Clément
2021/2022

Résumé

De nos jours, le tri des déchets est un problème important dans notre société, nous nous sommes alors donnés comme objectif de réaliser une station automatisée de tri des déchets. Les différents déchets sont tout d'abord acheminés sous une caméra par le biais d'un tapis roulant réalisé par nos soins à partir de rouleaux à pâtisseries et bandes de kinés. Cette caméra est reliée à un Raspberry Pi que l'on a doté d'un réseau de neurones, réseau que l'on a préalablement entrainé à reconnaitre différents déchets comme des masques ou des gobelets (cela peut évidemment s'étendre à d'autre objets). En fonction du déchet détecté, le Raspberry commande un servomoteur auquel nous avons attaché une longue plaquette de bois. Cette plaquette a pour but de faire dévier les différents déchets avançant sur le tapis vers leur poubelle respectives et ainsi accomplir le tri des déchets.

Summary

Nowadays, the sorting of waste is an important problem in our society, we then gave ourselves as objective to realize an automated station of sorting of waste. The different wastes are first of all routed under a camera by means of a conveyor belt realized by our care from pastry rollers and physiotherapy bands. This camera is connected to a Raspberry Pi that we have equipped with a neural network, a network that we have previously trained to recognize different waste such as masks or cups (this can obviously be extended to other objects). According to the detected waste, the Raspberry controls a servomotor to which we have attached a long wooden plate. The purpose of this plate is to divert the different waste advancing on the carpet to their respective garbage cans and thus accomplish the waste sorting.

Table des matières

Introduction

Le thème général du TIPE de cette année 2021-2022 étant « Santé-Prévention », nous avons décidé, après quelques semaines de réflexion, d'orienter nos travaux d'initiative personnelle encadrés sur le tri des déchets terrestres.

En effet, la mauvaise gestion des déchets, terrestres ou marins, contribue pour partie non seulement au changement climatique mais aussi à la pollution atmosphérique. Elle affecte directement des écosystèmes entiers et sera responsable, à terme, de la dégradation voire de la disparition d'une grande variété d'espèces.

De nombreuses actions peuvent permettre de réduire l'impact négatif de nos déchets sur l'environnement comme, par exemple, leur réduction directe ou leur recyclage, mais c'est sur leur tri et plus particulièrement sur le tri des déchets terrestres que nous avons choisi d'orienter nos travaux.

Manque de temps, indifférence, méconnaissance des règles de tri (quelle poubelle pour quel déchet ?) sont autant d'arguments avancés par les personnes qui négligent le tri de leurs déchets et préféraient de loin à ne pas avoir à s'en occuper.

C'est ainsi que nous est venue l'idée de réfléchir à un trieur de déchets.

Nous avons, dès le début de notre réflexion, souhaité matérialiser notre réalisation et nous nous sommes par conséquent fixé certaines contraintes, comme la réalisation d'un cahier des charges, pour atteindre notre objectif.

Après avoir choisi le milieu terrestre pour réaliser notre trieur de déchets, nous nous sommes interrogés sur la manière d'acheminer les déchets à notre trieur tout en réduisant au maximum l'intervention de l'homme afin de lui faciliter la tâche et donc lui être utile.

Nous avons tout d'abord envisagé de créer un robot qui ramasserait les objets au sol sur une certaine surface et procéderait de manière autonome à leur tri dans des poubelles adaptées en fonction de leur nature. L'intérêt nous est toutefois paru vite limité du fait de l'espace trop restreint de la zone traitée et car cela ne réglait pas le problème des personnes arguant de manque de temps ou de difficultés pour effectuer convenablement le tri.

Nous avons ensuite pensé à créer un tapis roulant sur lequel les déchets seraient automatiquement triés dès leur dépôt.

Nous avons supposé que ce trieur pourrait être installé dans un hôpital, les tris à effectuer étant particulièrement nombreux et réglementés (du fait de la nature même des déchets) au sein de ces établissements (seringues, verre, déchets radioactifs mais aussi cartons, restes alimentaires, …), mais notre système pourrait, en soit, être installé dans n'importe quelle entreprise qui génère des déchets.

Notre projet s'est naturellement scindé en trois grandes parties :

- la réalisation du tapis pour l'acheminement des déchets ;
- la détection et l'analyse des déchets ;
- le tri effectif des déchets.

I - Acheminement des déchets

Nous avions pensé au départ que les déchets seraient apportés par l'utilisateur au-dessus d'une poubelle compartimentée qui se décalerait en fonction du déchet dans le compartiment adapté, mais cette solution aurait été vite contestée par l'utilisateur peu enclin au tri du fait du temps passé à patienter au-dessus de la poubelle.

Nous avons donc, pour des raisons pratiques, rapidement opté pour l'acheminement des déchets à l'aide d'un tapis roulant.

Nous avions envisagé également certaines variantes susceptibles de rester en conformité avec la suite du projet : nous avions par exemple pensé à une sorte de petit train électrique qui aurait amené les déchets tel un tapis roulant. Le tri aurait alors pu s'effectuer en contrôlant les aiguillages et en utilisant des pistons ou un autre système pour faire tomber les déchets dans des poubelles lorsque le train passait à côté des bonnes poubelles.

Nous avons ensuite réfléchi à la façon de nous procurer ou de construire ce tapis roulant tout en modérant son coût et en déterminant nous-mêmes la quasi-totalité des paramètres de ce dernier (taille, largeur, vitesse, hauteur, poids, matière…)

Nous visualisions alors 3 types de tapis roulants :

- le tapis roulant, que l'on retrouve à la caisse des supermarchés ;
- le tapis de course ;
- et le tapis roulant destiné aux entrepôts de livraison dans l'industrie notamment ou en sortie de caisse c'est-à-dire celui en pente (comprenant beaucoup de rouleaux mis en contact les uns avec les autres).

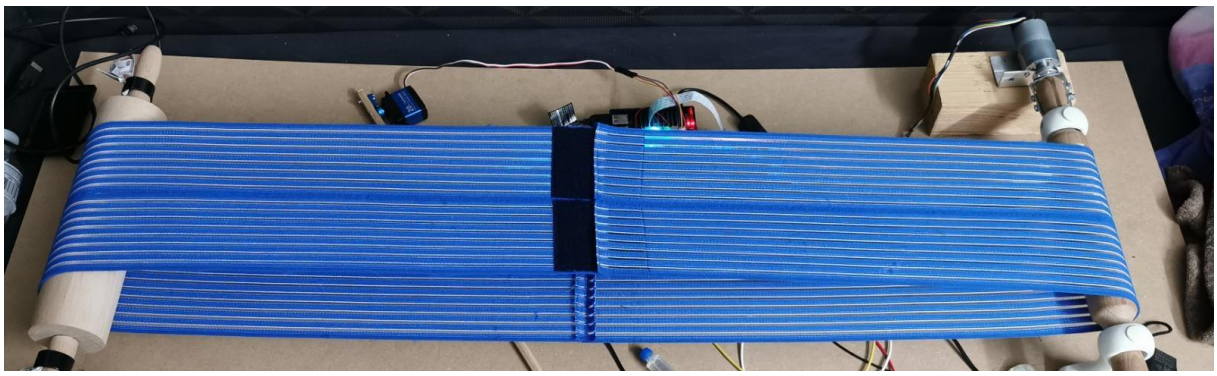
Nous avons alors compris qu'il nous fallait faire tourner le tapis roulant autour d'au minimum deux axes de rotations aux extrémités. Nous avons cherché quel objet utiliser pour cela : dans un premier temps c'est un tube PVC qui avait retenu notre attention car il est simple de se le procurer, on peut facilement choisir la longueur que l'on souhaite et cela n'est pas très cher. Dans un second temps et après réflexion nous avons trouvé plus pratique, moins cher et tout aussi simple de se procurer deux rouleaux à pâtisserie, un étant d'un bloc et l'autre étant monté sur un axe de rotation. L'avantage majeur que les rouleaux avaient par rapport aux tubes de PVC étaient leur manche qui pourront satisfaire plus facilement la mise en rotation.

Il fallait ensuite trouver l'objet qui allait nous servir de tapis : nous avons directement pensé (probablement à cause de la matière des tapis roulant classique) à une chambre à air de camion mais après quelque essais, nous nous sommes rendu compte que cela ne fonctionnerait pas : en effet, une

chambre à air est en forme de tore ainsi lorsque l'on découpe celle-ci de manière à obtenir un cylindre oblongue que l'on peut enrouler autour des rouleaux, on se rend compte qu'elle n'est pas assez plate (elle est légèrement bombée en son centre).

Nous avons ensuite pensé à un « tapis grillage » qui s'est rapidement avéré inadapté du fait des petits trous à travers desquels certains déchets auraient pu passer. Pour pallier ce problème, il aurait fallu le recouvrir de tissu mais cela perdait alors de son intérêt principal qui était la bonne accroche au rouleau. En effet, n'importe quel tissu ne convenait pas car il fallait que le tissu n'accroche pas les déchets mais accroche bien sur les rouleaux pour que ceux-ci ne tournent pas dans le vide et entraînent bien avec eux le tapis.

Nous avons finalement opté pour des bandes de kiné que nous avons cousues entre elles pour agrandir la largeur du tapis (deux bandes mises côte à côte) et en avons mis deux bout-à-bout pour agrandir également la longueur de celui-ci. Quatre bandes de kiné nous ont donc été nécessaires. Leur matière est parfaite car elle accroche bien aux rouleaux mais n'accroche pas les objets. Nous avons par ailleurs agrémenté notre construction de scratchs, ce qui nous permet de la monter et la démonter facilement.



Maintenant que notre tapis roulant est monté et fonctionne (tenu dans nos mains, mise en rotation avec nos mains) il a fallu réfléchir à la structure de celui-ci.

Trois options s'offraient alors à nous :

- des roulements à billes ;
- des supports de barres à rideaux ;
- ou deux bâtons en forme de Y.

La solution pour laquelle nous avons opté, est celle qui possédait le meilleur rapport qualité prix c'est-à-dire des supports de barres à rideaux. En effet, ceux-ci possèdent la forme des roulements à billes donc les frottements avec le manche du rouleau à pâtisserie sont plus faibles que si nous avions opté pour les deux bâtons en forme de Y. Les faibles frottements sont des arguments qu'il faut prendre en compte car cela permet d'avoir un moteur moins puissant pour une même vitesse de rotation.

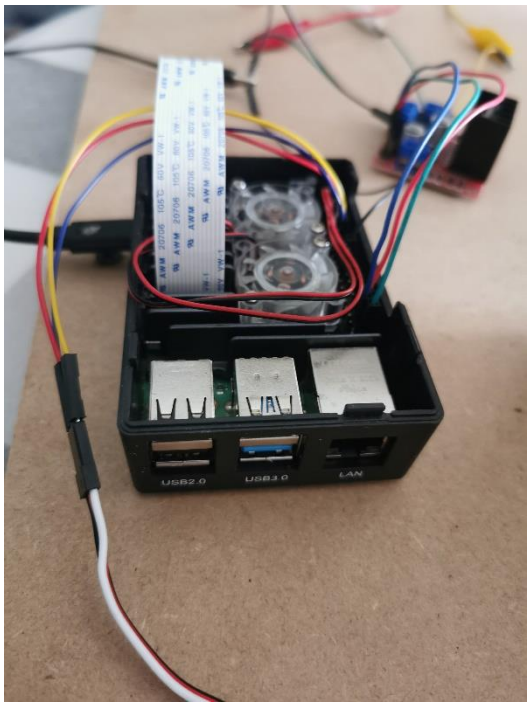
Pour finaliser la structure de notre tapis roulant nous avons pris des mesures sur une planche (de dimension 122.5 cm par 61 cm) de manière à fixer nos supports de barres à rideaux. Ainsi, il ne restait plus qu'à mettre en rotation un des rouleaux à pâtisserie pour que notre tapis tourne en continue de manière autonome (Nous développerons sa mise en rotation dans la partie Raspberry Pi). Pour cela, nous avons usiné une pièce de la forme du moteur, que l'on a fixé sur le manche du rouleau à pâtisserie constitué d'un bloc. Ainsi lorsque le moteur se met à tourner il entraine avec lui le rouleau et donc tout le tapis.

Nous nous sommes rendu compte d'un problème lorsque nous travaillons à faible vitesse. En effet, le tapis pouvait ralentir voire s'arrêter lorsque les scratchs arrivaient au niveau des rouleaux. Pour pallier ce problème, nous avons décidé de réduire la taille des scratchs.

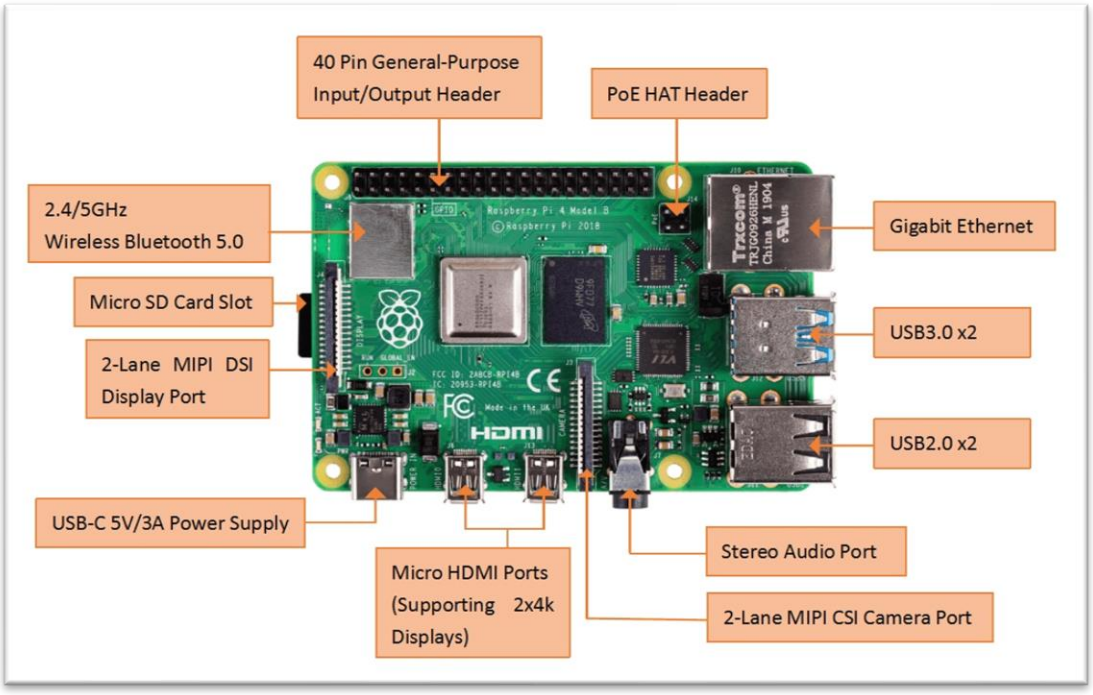
II - La détection et l'analyse des déchets

II.1. Le Raspberry Pi

Pour notre projet nous utilisons un Raspberry Pi 4 8Go pour contrôler le tout. Il s'agit d'un nano-ordinateur monocarte à processeur ARM de très petite taille, sorti en 2019. Il s'agit d'une évolution de la version précédente dont la plus ancienne a été créée en 2012. Nous avons choisi cette version car nous avions besoin d'un processeur suffisamment performant pour que notre réseau de neurones puisse fonctionner correctement. En effet, l'entraînement du réseau de neurones est un procédé qui demande beaucoup de ressources au processeur mais ce n'est pas le seul, l'analyse et le traitement du flux vidéo sont eux aussi importants.



Notre Raspberry Pi 4 8Go est équipé d'un processeur Broadcom BCM2711, quatre cœurs (Cortex-A72 64bit) ARM v8 cadencé à 1,5GHz, accompagné comme son nom l'indique de 8Go de RAM



Il possède 2 ports USB 2.0 et 2 ports USB 3.0, un port Ethernet Gigabit (900 Mb/s), un emplacement pour une carte micro SD (c'est sur celle-ci que nous allons mettre entre autres le système d'exploitation), 2 ports HDMI 4K, un port audio, un port pour une Pi Camera, un port d'alimentation USB Type C, les 40 broches GPIO (dont on détaillera l'importance lors de la connexion du moteur, du servomoteur et du ventilateur), un connecteur DSI (pour un écran de la marque Raspberry), une antenne wifi à 2.4GHz et le Bluetooth 5.0.

Lorsque l'on achète un Raspberry Pi, il est vierge, c'est-à-dire qu'il ne possède aucun système d'exploitation. Pour rappel un système d'exploitation est l'ensemble des programmes de base et des utilitaires qui permettent à l'appareil informatique de fonctionner. C'est donc à nous de choisir lequel nous souhaitons installer. Nous avons alors fait le choix de mettre un système dit « libre » c'est-à-dire un GNU/Linux, plus précisément le nom de notre système est Debian GNU/Linux 10 (buster). Il s'agit-là de la dixième version de ce système d'exploitation. C'est d'ailleurs la distribution la plus installée sur un Raspberry Pi car elle a été créée pour ce genre de système. On dit de ce système qu'il est libre car c'est un système dont le code est « open source », autrement dit n'importe qui peut, s'il le souhaite avoir accès au code source du système d'exploitation et le modifier à sa guise.

D'autres distributions de GNU/Linux étaient disponibles, comme par exemple, Slackware (une des plus anciennes distributions de Linux), Mandriva (nous avions commencé par ce système d'exploitation car nous avions vu qu'il était simple d'utilisation mais sa simplicité nous restreignait, nous avons donc abandonné cette option), Red Hat mais nous ne les avons pas sélectionnés.

Maintenant que nous avons installé le système d'exploitation sur notre Raspberry Pi, nous allons pouvoir commencer à l'utiliser.

Il existe deux façons d'utiliser un système GNU/Linux :

- Le mode console : c'est le mode que l'on a principalement utilisé (toutes les interactions que l'on fait avec notre Raspberry Pi se font via un terminal de commande)

- Le mode graphique : c'est est un mode où l'on dispose d'un bureau, similaire au bureau de Windows, la souris fonctionne, il y a des icônes.

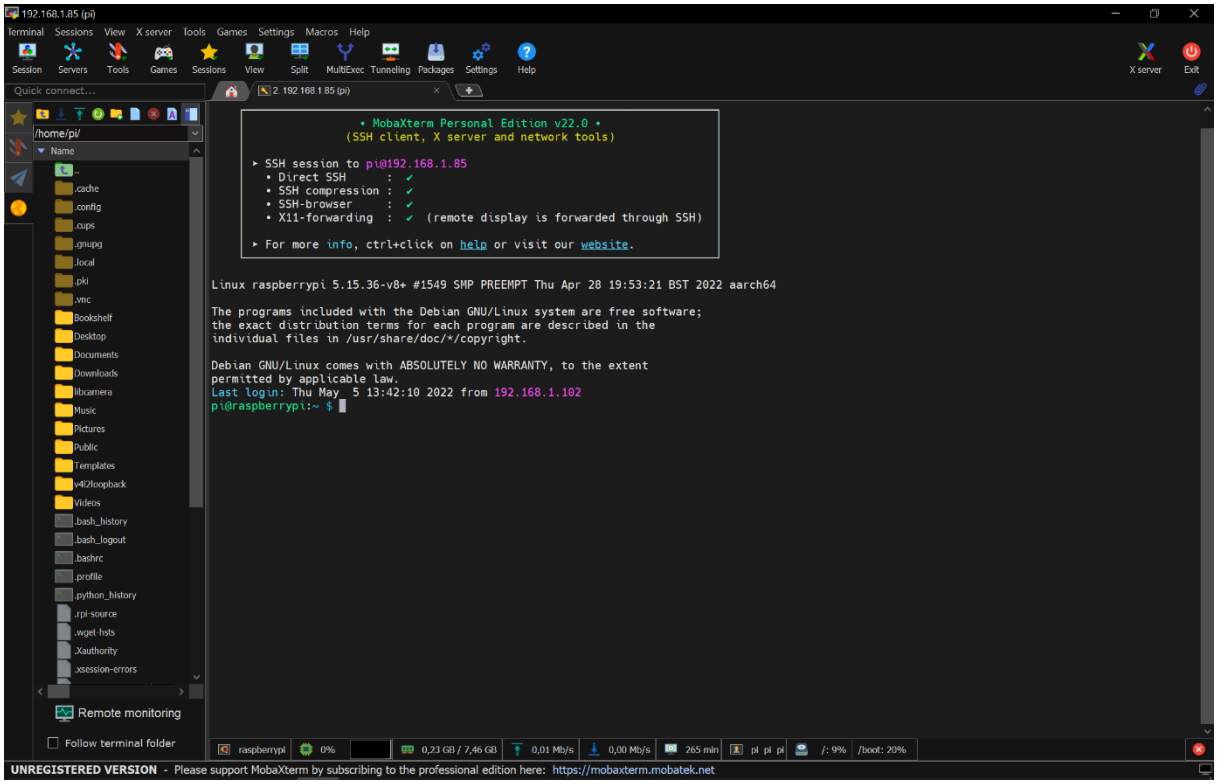
Nous avons tout d'abord branché à un écran, un clavier et une souris de manière à pouvoir régler les premiers paramètres (mode graphique).

Puis, pour être plus pratique, nous avons changé la langue du clavier, nous sommes passés d'un clavier QWERTY en un clavier AZERTY.

Ensuite, il était important de modifier les options de démarrage, l'identifiant et le mot de passe d'origine. Nous avons aussi mis à jour l'ensemble de notre v2 GNU/Linux (chose à faire régulièrement sous environnement Linux).

Enfin, nous avons activé le port Caméra pour pouvoir utiliser une Pi Caméra et la connexion SSH pour pouvoir se connecter à distance sur le Raspberry PI. Ce second point est très important car il nous permet d'utiliser notre ordinateur pour se connecter au Raspberry PI. Ainsi, nous n'avons plus besoin de transporter un écran, un clavier et une souris car nous pouvons utiliser celui de notre ordinateur.

Une fois le port activé, il faut se connecter au terminal via le service SSH, qui permet une connexion à distance et sécurisée. Cette connexion se fait via MobaXTerm, qui est un logiciel qui a une interface agréable et permet de se connecter facilement à distance sur des appareils comme le Raspberry Pi, en renseignant son IP et son port et en fournissant le nom d'utilisateur et le mot de passe du compte créé

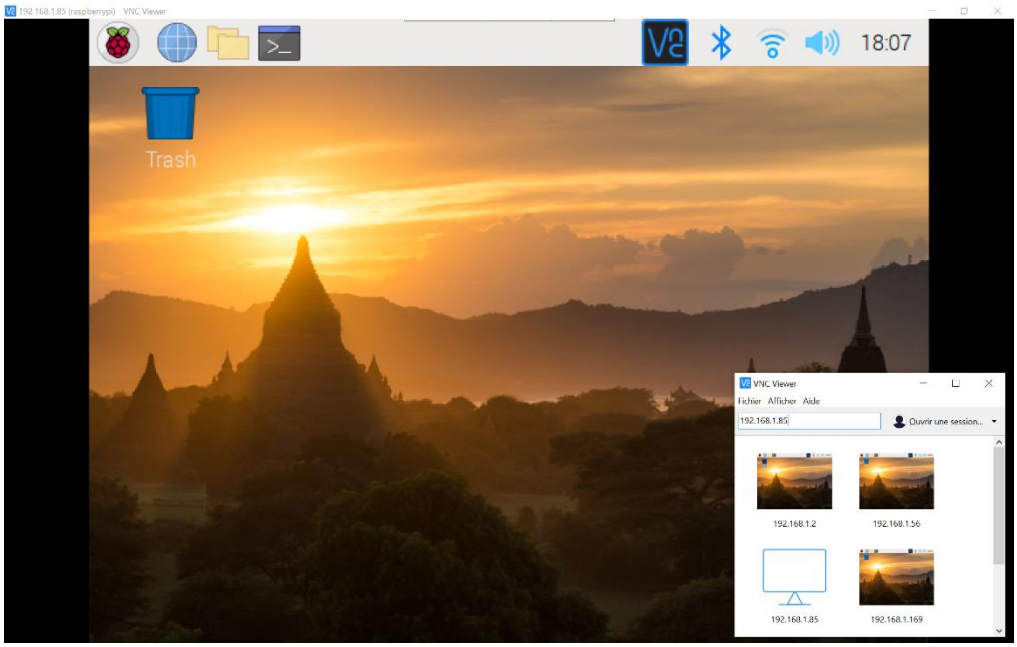


sur le Raspberry Pi. Après s'être connecté, le Raspberry PI peut être géré à distance et des fichiers peuvent être transférés via rsync (commande de transfert de fichier sur Linux).

Une fois tous ces paramètres rentrés, le logiciel nous propose alors un terminal, avec sur la gauche 'les dossiers de notre nano-ordinateur' et sur la droite une console dans laquelle nous allons rentrer toutes nos commandes (mode console).

Pour pouvoir se connecter en local, mais aussi à distance, il était nécessaire d'ouvrir les ports de notre box, pour cela, il a fallu se connecter au site de notre opérateur et ouvrir un port pour notre Raspberry Pi.

Parfois, il était plus simple d'avoir un vrai bureau et non un terminal de commande pour se connecter au Raspberry Pi (mode graphique), ainsi nous avons aussi appris à utiliser le logiciel VNC Viewer qui nous donne alors le même visuel que si nous branchions un écran directement au Raspberry Pi



Pour ces deux logiciels, il est nécessaire de récupérer l'IP de notre Raspberry Pi pour ensuite pouvoir s'y connecter, habituellement cette IP est fixe mais pas dans notre cas. Nous avons donc essayé de résoudre ce problème mais sans succès. Nous avons donc dû le contourner, pour cela nous avons utilisé le logiciel : Advanced IP Scanner qui nous permet d'analyser le réseau c'est-à-dire afficher toutes les IP des appareils connectés à celui-ci et donc de retrouver l'IP du Raspberry Pi à partir du moment où celui-ci y est connecter. Pour connecter notre Raspberry Pi au réseau, nous devons d'abord, le connecter par câble Ethernet pour ensuite pouvoir s'y connecter avec VNC Viewer et seulement après, pouvoir accéder (en haut à droite de l'écran) aux différents réseaux wifi. Une fois connecté à celui souhaité, nous pouvons débrancher notre câble Ethernet et une nouvelle fois en utilisant Advanced IP Scanner retrouver l'IP de notre Raspberry Pi sur le nouveau réseau.

Après toutes ces étapes pour paramétrer le Raspberry Pi, nous pouvions désormais commencer à installer les modules qui serviront à exécuter notre programme de reconnaissance d'objet et notre algorithme de tri. Nous avons donc commencé par installer python 3, pour pouvoir depuis le terminal de commande, exécuter les fichiers python que l'on déposait sur le Raspberry Pi (à l'aide de la commande « python3 nom_du_fichier.py »).

En général, la plupart des fichiers pythons utilisent des bibliothèques. Ainsi, pour rendre nos fichiers exécutables sur notre Raspberry Pi, il nous a été nécessaire de les installer. Pour cela, nous avons utilisé la commande suivante : « pip install nom_du_module ».

Nous cherchons désormais à récupérer le flux vidéo de notre caméra. Pour cela, dans un premier temps, nous avons utilisé plusieurs modules, tout d'abord le module libecamera avec ses options comme preview pour savoir si celle-ci fonctionnait. Dans un deuxième temps, nous avons utilisé le module picamera mais celui-ci ne fonctionnait pas. Après avoir analysé plusieurs erreurs et corrigé beaucoup d'entre-elles, nous nous sommes rendu compte que cela ne fonctionnerait pas à cause de la version du système d'exploitation installée sur notre Raspberry Pi. Ainsi, nous avons dû trouver une autre solution. Nous avons alors été obligés d'abandonner notre Pi camera, pour une caméra que nous pouvons brancher en USB, cela nous permet alors d'utiliser les commandes de pilotage de caméra prédéfinie dans le module OpenCV. Ce module nous est familier car nous l'avons beaucoup utilisé dans notre TIPE de première année mais aussi cette année en informatique avec le chapitre Traitement d'image ou encore dans la création du réseau de neurones.

Nous disposions alors d'un flux continue d'images provenant de notre caméra. Il fallait donc désormais l'analyser de manière à reconnaître les différents déchets. Pour cela, nous avons créé notre propre réseau de neurones.

II.2. Le réseau de neurones

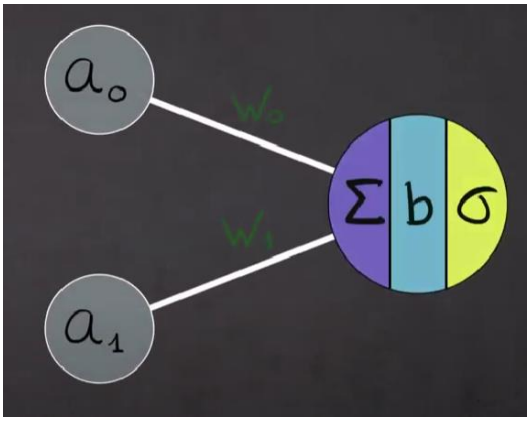
Afin de trier les déchets, notre station doit être capable de les reconnaître. Pour cela nous avons décidé de la doter d'un réseau de neurones.

Un réseau de neurones artificiels est un système informatique s'inspirant du fonctionnement du cerveau humain pour apprendre. Par le biais d'un algorithme, le réseau de neurones artificiels permet à l'ordinateur d'apprendre à partir de nouvelles données. Le système doté du réseau de neurones apprend à effectuer une tâche en analysant des exemples pour s'entraîner.

II.2.1. Le Deep Learning

Ce qui caractérise le « Deep Learning » par rapport à toutes les formes d'intelligences artificielles c'est qu'on se contente de lui donner un but à accomplir et qu'il va apprendre de lui-même comment l'atteindre. Le principe va être d'utiliser des réseaux de neurones qui vont interagir entre eux pour s'adapter à un problème donné. Par exemple, si nous devions programmer un algorithme standard capable de différencier un chiffre écrit à la main d'un autre, il faudrait d'abord définir des règles pour lui permettre de détecter les lignes, les angles, les courbes, pour ensuite lui enseigner comment sont composées les différents chiffres. Pour le « Deep learning », c'est radicalement différent. Le principe va être de fournir un très grand nombre d'exemples au réseau de neurones, suffisamment pour qu'il puisse de lui-même apprendre comment s'adapter à notre problème. Dans l'exemple de nos chiffres, il faudra lui donner des centaines, voire des milliers d'images de différents chiffres, pour qu'il arrive à identifier quels sont les paramètres communs entre toutes ces images. Au début il sera plutôt nul à son job mais, au fur et à mesure des exemples il s'améliorera.

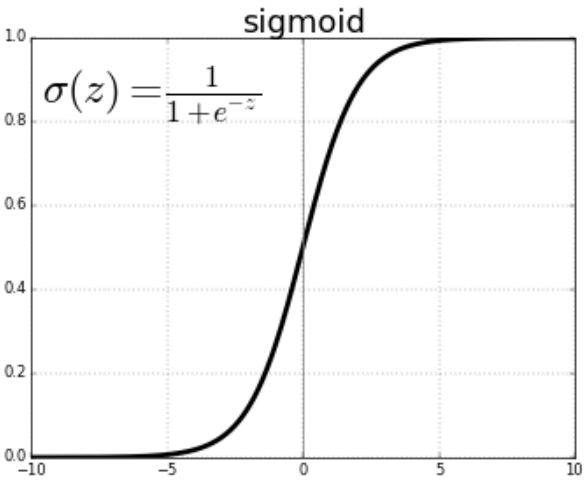
On va tenter de définir comment un réseau de neurones peut apprendre à résoudre un problème donné. Notre réseau de neurones est donc un ensemble de neurones organisés en couches et reliés entre eux par des connexions qu'on appelle des poids. Chacun de ces poids est une valeur qui nous servira à propager nos informations. Ils sont à la base du fonctionnement de notre réseau ; c'est à partir d'eux que tout l'apprentissage pourra se faire. La première couche de neurones d'entrée s'appelle le « input layer ». Notre couche de neurones de sortie, le « output layer » et toutes les couches qui se trouvent entre les deux portent le nom de « hidden layers ». Le nombre de « hidden layers » est très variable, potentiellement de 0 à l'infini. Cependant, nous parlerons le « Deep Learning » pour les réseaux de neurones qui ont au moins 2 « hidden layers » (c'est à partir de ce nombre de couches que notre réseau sera capable de traiter des problèmes vraiment complexes).



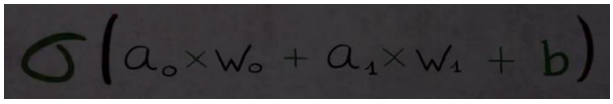
Commençons par nous intéresser au fonctionnement du plus petit réseau de neurones possible : un unique neurone avec 2 entrées. Le neurone reçoit plusieurs valeurs d'entrée (a0 et a1 sur l'image) et produit une valeur de sortie. Chacune de ses entrées est pondérées par un poids (w0 et w1 sur l'image). A l'intérieur de ce neurone il y a une fonction d'activation (σ sur l'image),

elle va nous servir à faire rentrer nos valeurs d'entrée dans un petit intervalle, le plus souvent entre 0 et 1 ou alors entre moins -1 et 1. Les fonctions les plus utilisées sont la tangente hyperbolique, la sigmoïde, la rectification linéaire.

Dans notre exemple, nous allons utiliser la fonction sigmoïde, dont voici l'expression et la courbe ci-dessous, qui nous renverra un résultat entre 0 et 1.



Chaque neurone fonctionne en 3 temps : nous commençons par faire une moyenne pondérée de ses entrées qu'on multiplie par leur poids correspondant. Nous ajoutons ensuite ce qu'on appelle un biais, c'est une valeur caractéristique à chaque neurone et enfin nous nous retrouvons avec notre résultat qu'on peut envoyer à notre fonction d'activation.

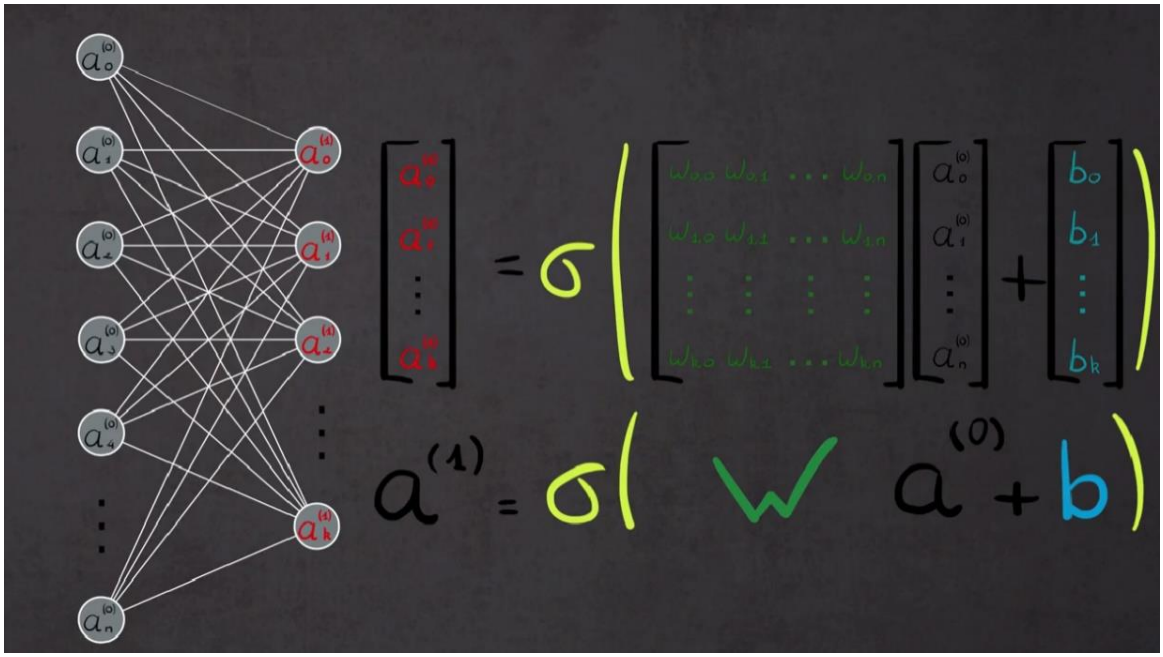


Équation finale

Plus la valeur que nous lui donnons sera grande, plus la sortie se rapproche de 1 et à l'inverse, plus la valeur d'entrée sera négative plus la sortie se rapproche de 0. C'est ce résultat de sortie qui nous dira à quel point notre neurone est activé, à 0 nous dirons de notre neurone qu'il est totalement inactif, à un il sera totalement activé ; et il peut avoir tous les États intermédiaires et c'est là que, pour augmenter notre contrôle sur cette valeur de sortie, nous allons faire intervenir le biais. C'est en quelque sorte le seuil au-delà duquel nous considérerons notre neurone comme actif : par exemple, si actuellement notre fonction sigmoïde nous retourne un neurone activé à 50 pour cents, quand elle reçoit 0 en entrée, si nous rajoutons à notre équation un grand biais, notre neurone sera activé beaucoup plus facilement. Cette valeur nous permet en quelque sorte de décaler la fonction d'activation. Nous allons donc rajouter le biais à la valeur d'entrée de notre fonction. Pendant la phase d'entraînement de notre réseau, cette valeur sera ajustée en même temps que tous nos poids. Ce processus pendant lequel nos valeurs se propagent jusqu'à nous retourner un résultat s'appelle le « feed forward ».

Nous avons donc le fonctionnement d'un neurone. Mais dans le cas de notre réseau il y a des centaines. Chacun avec un grand nombre de connexions et au total nous nous retrouvons avec plus d'un million de paramètres, et c'est tout le but de notre apprentissage : trouver les bonnes valeurs pour chacun d'entre eux de manière à ce que chaque déchet soit correctement reconnu. A la création du réseau, nous initialisons tous nos poids et biais avec des valeurs aléatoires et à chaque tour de « feed forward » ils vont être progressivement modifiés pour que le résultat du réseau de neurones soit de plus en plus pertinent.

Dans notre exemple pour la reconnaissance de déchets, il y a plusieurs couches et des centaines de neurones et il devient donc impossible de noter une formule avec autant de paramètres. Nous allons donc lui donner une notation plus compacte en utilisant les matrices et nous voit si dessous ce a quoi la nouvelle formule va ressembler.



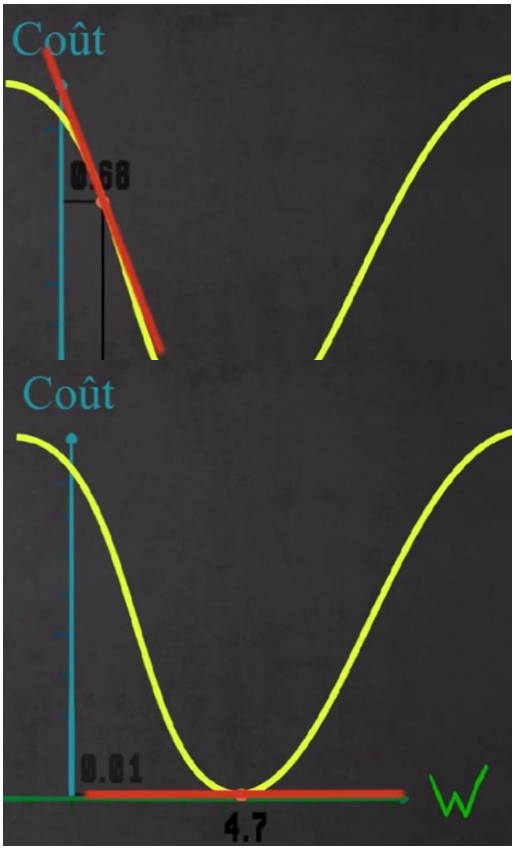
A gauche de l'image nous voyons à quoi ressemble les 2 premières couches et nous pouvons noter chaque neurone de notre première couche avec un « a », en indiquant le numéro de la ligne en indice et le numéro de la couche en exposant. Ainsi « $a_3^{(0)}$ » est le 3e neurone de la 1e couche (on considère que 0 est la première couche). De plus, nous notons « $w_{0,k}$ », le poids qui relie le k^{ieme} neurone de la première couche au premier neurone de la 2e couche, c'est-à-dire le poids qui relie « $a_k^{(0)}$ » à « $a_0^{(1)}$ ». Ils constitueront la première ligne de la matrice **W** des poids. Nous pouvons remplir les lignes suivantes en utilisant le même principe avec tous les autres poids.

Le fait de remplir cette matrice avec tous les poids qui relient la première et la 2e couche, nous permettra d'obtenir le vecteur résultat de la 2e couche de neurones en un seul calcul. Le résultat sera un vecteur avec une valeur pour chaque neurone de notre 2e couche. Nous initialisons ensuite le vecteur des biais qui contient les valeurs des biais de tous les neurones. Une fois toutes les matrices initialisés notre calcul ressemble ça :

Pour finir, il nous reste à passer ce résultat à travers notre fonction d'activation, c'est ce que l'on retrouve en bas à droite de l'image plus haut. Dans ce calcul, « $a^{(1)}$ » est, comme nous l'avons dit, le vecteur qui représente le taux d'activation des neurones de la 2e couche, il contient exclusivement des nombres compris entre 0 et 1. Pour terminer d'initialiser notre réseau il ne nous reste plus qu'à appliquer cette formule pour les couches restantes jusqu'à notre « ouput layer ».

Désormais, toutes nos valeurs sont initialisées, nous avons en particulier accès à nos valeurs de sortie qui sont présentes dans notre « ouput layer », il va falloir permettre à notre réseau d'apprendre. Pour cela, nous allons commencer à calculer les progrès qui lui reste à faire à la création de notre réseau. Quand nous lui donnons une première image, le résultat va être aléatoire. Les neurones de sortie vont alors tous être plus ou moins activés, tandis que nous attendons que seul le neurone du déchet présent sur l'image soit activé. Pour savoir à quel point notre réseau doit être modifié, nous devons calculer la différence entre la valeur de chacun des neurones de sortie de notre réseau et la valeur qu'il est censé nous donner. Nous nous retrouvons avec le coût de cette image. Nous allons nous intéresser au calcul du coût d'un très grand nombre d'échantillons d'images pour en faire une moyenne. Cette valeur est, comme nous l'avons déjà dit, une représentation du niveau d'erreur de notre réseau pour chacun de ces neurones de sortie. Ensuite, parmi nos paramètres, il va falloir identifier quelle combinaison de poids et de biais nous permettra d'avoir le coût le plus petit possible. Pour cela nous utilisons un algorithme connu sous le nom de : descente du gradient. Cet algorithme consiste à étudier le coût en fonction des poids et des biais et de trouver le coût minimum.

Imaginons un instant qu'il n'y ait qu'un seul poids à modifier pour réduire notre coût et représentons cela sur un graphique en 2 dimensions, avec la valeur de notre poids en abscisse et le coût qu'il nous renvoie en ordonnée. Émettons l'hypothèse que notre fonction de coût sera représentée comme ci-contre (courbe en jaune) :



Ensuite, nous allons entrer une valeur totalement aléatoire et, plutôt que de tester toutes les autres valeurs, nous allons utiliser les dérivés ou plus précisément, l'information que nous procure la dérivée par rapport à la pente de la tangente à la courbe en un point donnée. En effet, en fonction de cette tangente, nous serons en mesure de déterminer comment changer le poids en entrée pour se rapprocher du résultat recherché : nous rappelons quand même que le but ici est d'avoir le coût le plus petit possible. Si la dérivée est positive, nous devrons diminuer notre poids, et inversement. Également, plus cette dérivée est importante, plus nous sommes loin du résultat attendu,

nous allons alors pouvoir augmenter notre « input », re calculer la dérivée et recommencer ces étapes encore et encore jusqu'à atteindre une dérivée nulle qui signifiera qu'on atteint le coût minimum et donc le meilleur résultat possible.

Pour notre réseau de neurones nous appliquons le même principe, mais avec des milliers de paramètres à modifier pour trouver le coût minimum. En utilisant la descente de gradient couche par couche, nous corrigeons ainsi la valeur des poids et des biais qui ont conduit à l'erreur : c'est le procédé de back propagation.

Et voilà, il ne nous reste plus qu'à reproduire cette étape de « feed forward » et de back propagation encore et encore jusqu'à avoir notre résultat optimal.

II.2.2. Réalisation du réseau de neurones

II.2.2.1. Présentation des outils

Pour coder notre réseau de neurones sur python, nous utiliserons TensorFlow et Keras. TensorFlow est une bibliothèque open source créer pour Python. TensorFlow compile de nombreux algorithmes et modèles différents, permettant à l'utilisateur de mettre en œuvre des réseaux de neurones à utiliser dans des tâches telles que la reconnaissance/classification d'images.

Keras est un API (Application Programming Interface) qui peut utiliser les fonctions de TensorFlow. Keras a été conçu avec la convivialité et la modularité comme principes directeurs. Ainsi, Keras rend la mise en œuvre des nombreuses fonctions puissantes, mais souvent complexes de TensorFlow, aussi simple que possible. Il est configuré pour fonctionner avec python sans aucune modification ou configuration majeure.

II.2.2.2. Classification d'image et CNN

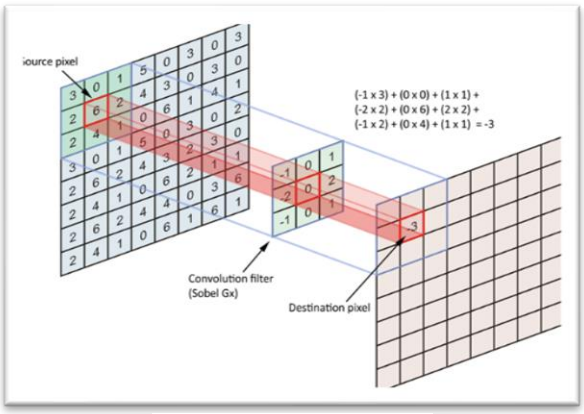
La classification d'images fait référence à la tâche d'entrer une image dans un réseau de neurones et de lui faire produire une sorte d'étiquette pour cette image. L'étiquette que le réseau produit correspondra à une classe prédéfinie, dans notre cas ces classes seront le nom de nos déchets.

Pour effectuer la classification d'images, nous utilisons un type de réseau de neurones particulier : le réseau neuronal convolutif (CNN). Ces réseaux possèdent une architecture propre. En effet, contrairement à un modèle classique qui ne contient qu'une partie classification, l'architecture du CNN dispose en amont d'une partie convolutive. L'objectif de cette partie est d'extraire des caractéristiques propres à chaque image en les compressant de façon à réduire leur taille initiale.

II.2.2.1.1. Partie convolutive

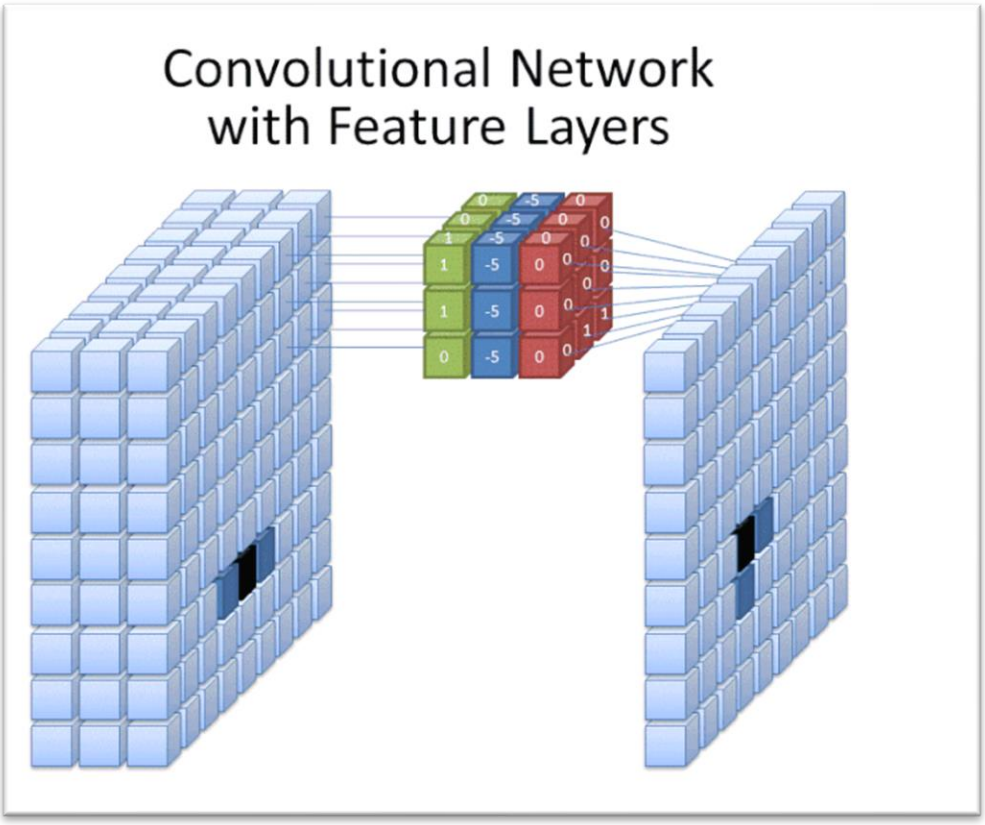
Dans un premier temps, le réseau de neurones doit procéder à l'extraction de caractéristiques. Les caractéristiques sont les éléments des données qui vous intéressent et qui seront introduits dans le réseau. Dans le cas spécifique de la reconnaissance d'images, les caractéristiques sont les groupes de pixels, tels que les bords et les points, d'un objet que le réseau analysera à la recherche de modèles. Ce processus d'extraction des caractéristiques d'une image est réalisé à l'aide d'une "couche convolutionnelle". La convolution est le traitement d'une matrice par une autre appelée matrice de convolution ou « noyau ». Notre filtre « matrice de convolution » utilise une première matrice qui est l'image, c'est-à-dire une collection de pixels en coordonnées rectangulaires 2D, et un noyau que l'on choisit ici de taille 3x3. Le filtre étudie successivement chacun des pixels de l'image. Pour chaque pixel, que nous appellerons « pixel source » (comme sur l'image ci-dessous), il multiplie la valeur de ce pixel et de chacun des 8 pixels qui l'entoure par la valeur correspondante dans le noyau. Il additionne l'ensemble des résultats et le pixel de la nouvelle matrice ayant les mêmes coordonnées que le pixel source (Distribution pixel sur l'image) prend alors la valeur du résultat final.

Le filtre parcourt toute la matrice principale de manière incrémentale et génère une nouvelle matrice en reproduisant le processus expliqué plus haut.



Processus de convolution

Une taille de filtre courante (et celle que l'on utilisera) utilisé dans les CNN est 3, et cela couvre à la fois la hauteur et la largeur, de sorte que le filtre examine une zone de 3x3. Cependant la profondeur du filtre doit également être spécifiée. Les images numériques sont rendues sous la forme d'une hauteur, d'une largeur et d'une valeur RVB qui définit les couleurs du pixel. La "profondeur" qui est suivie est donc le nombre de canaux de couleur de l'image. Par exemple, les images en niveaux de gris n'ont qu'un seul canal de couleur tandis que les images en couleur ont 3 canaux de couleurs. Ainsi, pour un filtre de taille 3 appliqué à une image en couleur, les dimensions de ce filtre seront de 3 x 3 x 3. Le processus de convolution avec un filtre en trois dimensions est assez similaire à celui à deux dimensions expliqué plus haut : pour chaque couche de l'image, correspondant aux couches de rouge, vert et bleu, nous réalisons le processus de convolution en deux dimensions puis nous additionnons les trois résultats pour obtenir la valeur du nouveau pixel.



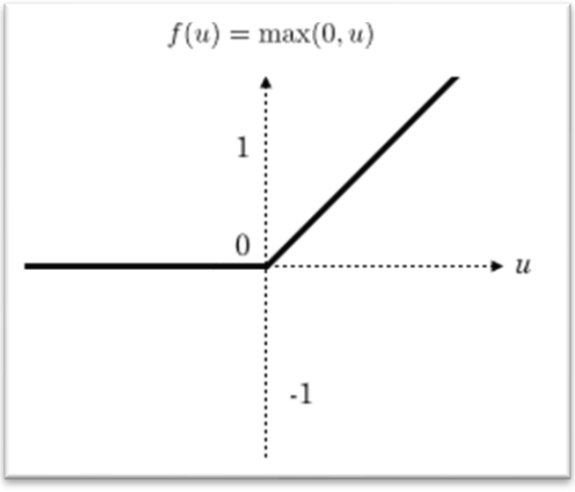
Ce processus est ensuite effectué pour l'image entière afin d'obtenir une représentation complète.

Il est généralement effectué avec plus d'un filtre, ce qui permet de préserver la complexité de l'image.

Le résultat final de tous ces calculs est une carte des caractéristiques.

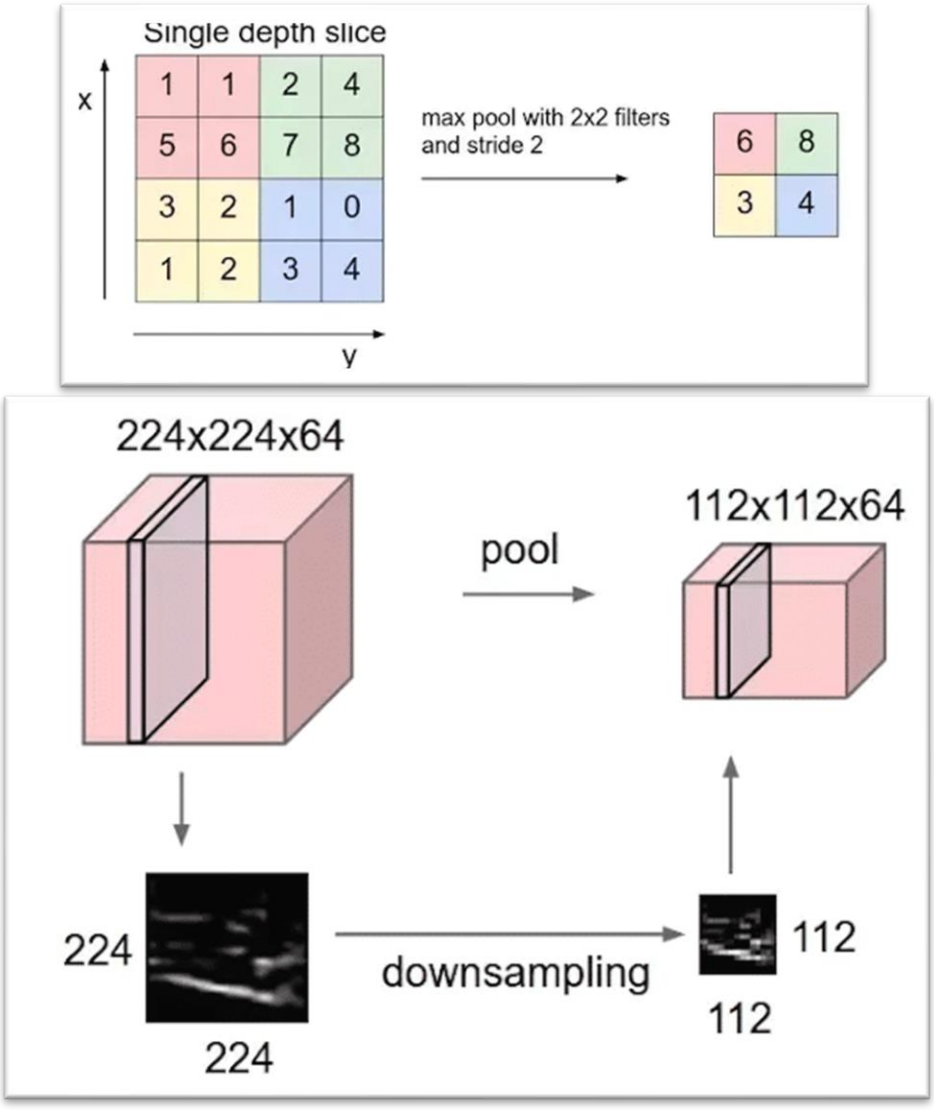
Après la création de la carte de caractéristiques de l'image, les valeurs qui représentent l'image passent par une fonction d'activation ou une couche d'activation. La fonction d'activation prend les valeurs qui représentent l'image, qui sont sous une forme linéaire (c'est-à-dire juste une liste de nombres) grâce à la couche convolutionnelle, et augmente leur non-linéarité puisque les images elles-mêmes sont non linéaires.

La fonction d'activation typique utilisée à cette fin est une unité linéaire rectifiée (ReLU). Cette fonction, dont voici la courbe ci-contre, remplace toutes les valeurs négatives reçues en entrées par des 0. L'intérêt de cette fonction est de rendre le modèle non linéaire et de ce fait plus complexe.



Une fois les données activées, elles sont envoyées à travers une couche de mise en commun. La mise en commun « sous-échantillonne » une image, ce qui signifie qu'elle prend les informations qui représentent l'image et les compresse, les rendant plus petites. Le processus de mise en commun rend le réseau plus flexible et plus apte à reconnaître des objets ou des images sur la base des caractéristiques pertinentes. La méthode la plus utilisée de sous-échantillonnage est le Max-Pooling : le Max-Pooling est un processus de discrétisation basé sur des échantillons. Son objectif est de sous-échantillonner une représentation d'entrée en réduisant sa dimension. De plus, son intérêt est qu'il réduit le coût de calcul en réduisant le nombre de paramètres à apprendre et fournit une invariance par petites translations (si une petite translation ne modifie pas le maximum de la région balayée, le maximum de chaque région restera le même et donc la nouvelle matrice créée restera identique).

Pour rendre plus concret l'action du Max-Pooling, voici un exemple : imaginons que nous avons une matrice 4×4 représentant notre entrée initiale et un filtre d'une fenêtre de taille 2×2 que nous appliquerons sur notre entrée. Pour chacune des régions balayées par le filtre, le Max-Pooling prendra la valeur de pixels maximale, créant ainsi par la même occasion une nouvelle matrice de sortie où chaque élément correspondra aux maximums de chaque région rencontrée.



Exemple d'effet de Max-pooling

La fenêtre de filtre se déplace de deux pixels vers la droite (stride/pas = 2) et récupère à chaque pas “l’argmax” correspondant à la valeur la plus grande parmi les 4 valeurs de pixels. Ainsi la partie convolutive d’un CNN permet d’obtenir en sortie une « carte de caractéristiques » ou « code CNN » dont les dimensions sont plus petites que celles de l’image initiale ce qui va avoir l’avantage de diminuer grandement le nombre de paramètres à calculer dans le modèle. La partie convolutive est très souvent constituée d’un enchainement de couches de convolution et de couches de pooling.

II.2.2.1.2. Partie classification

Le code CNN obtenu en sortie de la partie convolutive est fourni en entrée dans une deuxième partie, constituée de couches entièrement connectées appelées perceptron multicouches (MLP pour Multi Layers Perceptron) dont le fonctionnement est expliqué dans la partie Le principe théorique.

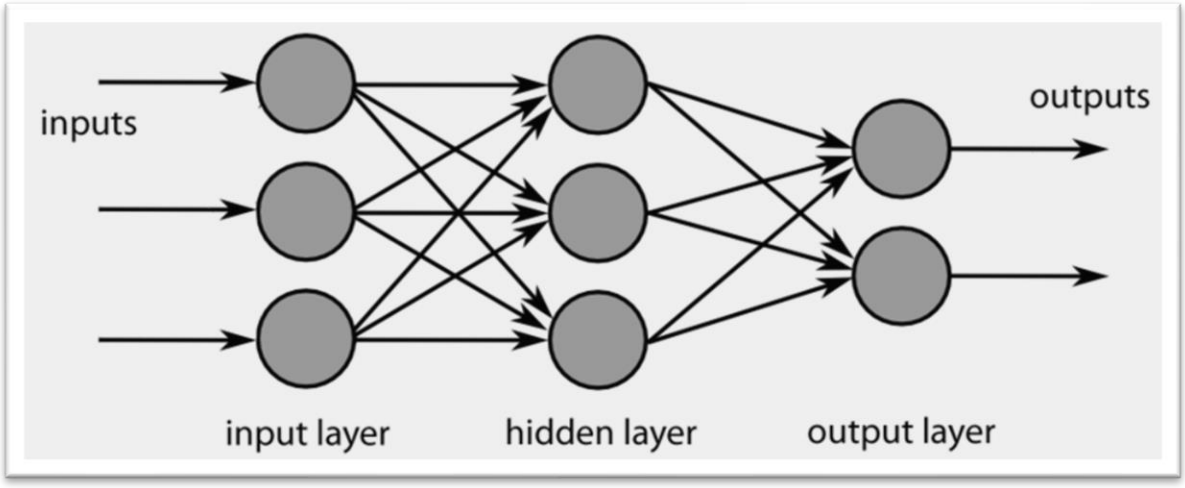


Schéma d'un réseau de neurones artificiels multicouches

Cependant, pour que ces données soient traitées, elles doivent être sous la forme d’un seul vecteur. Les données doivent donc être « aplaties », c’est-à-dire compressées dans un long vecteur avant d’arriver en entrée de la seconde partie. Le rôle de cette dernière est de combiner les caractéristiques du code CNN afin de classer l’image. Elle renvoie en sortie un vecteur de taille d correspondant au nombre de classes dans lequel chaque composante représente la probabilité pour l’input image d’appartenir à une classe.

II.2.2.3. Implémentation du réseau en python

II.2.2.3.1. Import des modules et des données

Après avoir importé les modules nécessaires de TensorFlow et de Keras, nous devons rassembler les données nécessaires à l'entrainement de notre réseau.

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

#
img_height = 180
img_width = 180
batch_size = 32

train_ds = tf.keras.utils.image_dataset_from_directory('resources/dechets/', validation_split=0.2, subset="training",
                                                       seed=123, image_size=(img_height, img_width),
                                                       batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory('resources/dechets/', validation_split=0.2, subset="validation",
                                                     seed=123,
                                                     image_size=(img_height, img_width), batch_size=batch_size)


class_names = train_ds.class_names
```

Pour cela nous utilisons la fonction tf.keras.utils.image_dataset_from_directory qui permet de passer d'un répertoire d'images sur un disque à un objet de type « dataset », utilisé pour le réseau de neurone.

Dans les deux commandes ci-dessus nous retrouvons :

- -'ressources/déchets/' : c'est le dossier dans lequel sont les données
- -validation_split=0.2 : ceci correspond à la proportion de données que l'on utilise pour la validation, nous utiliserons ici 80% des images pour la formation et 20% pour la validation.
- -subset='training' ou subset='validation' : nous précisons ici dans quelle partie du dataset l'on se trouve : partie d'entrainement ou partie validation.
- -seed=123 : graine aléatoire facultative pour le brassage et les transformations.
- -image_size=(img_height, img_width) : taille à laquelle les images sont redimensionnées après avoir été lues sur le disque.
- -Batch_size=batch_size : le batchsize définit le nombre d'échantillons qui vont être propagés à travers le réseau.

On récupère finalement le nom de nos classes qui seront « masque » et « gobelet » et « boite ».

Nous pouvons visualiser quelques images présentes dans le dataset :

Au total nous disposons de près de 700 photos qui sont reparties équitablement entre les 3 classes.

On doit ensuite configurer nos données.

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

- Dataset.cache conserve les images en mémoire après leur chargement hors disque au cours de la première époque. Cela garantira que l'ensemble de données ne devienne pas un goulot d'étranglement lors de la formation de votre modèle. Si votre jeu de données est trop volumineux pour tenir en mémoire, vous pouvez également utiliser cette méthode pour créer un cache sur disque performant.

- Shuffle mélange aléatoirement les éléments de cet ensemble de données. L'argument **1000** représente le nombre d'éléments de ce jeu de données à partir duquel le nouveau jeu de données sera échantillonné.

- Dataset.prefetch permet de préparer les éléments ultérieurs pendant que l'élément actuel est traité. Cela améliore souvent la latence et le débit, au prix de l'utilisation de mémoire supplémentaire pour stocker les éléments pré-extraits. L'argument **buffer size** représentant le nombre maximum d'éléments qui seront mis en mémoire tampon lors de la pré-extraction. La valeur tf.data.AUTOTUNE est utilisée afin que la taille du tampon soit ajustée dynamiquement.

Une dernière étape pour le prétraitement des données est de les standardiser. En effet, les valeurs des canaux RVB sont dans l'intervalle [0, 255] ce qui n'est pas idéal pour un réseau de neurones, nous préférons avoir des valeurs plus petites. Dans notre cas nous allons normaliser nos valeurs pour qu'elles soient toutes dans l'intervalle [0, 1]. Pour cela nous introduirons une couche supplémentaire au début de notre réseau qui normalisera nos données.

II.2.2.3.2 Création du réseau

On peut maintenant s'attaquer à la création du modèle. Voilà à quoi cela ressemble :

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Conv2D(256, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.3),
  layers.Dense(num_classes)
])
```

On retrouve bien la couche Rescaling qui normalise nos données ainsi que les différentes couches de convolution et de pooling introduites dans la partie Classification d'image et CNN. Les arguments de la fonction Conv2D sont :

- La dimension de l'espace de sortie, c'est le premier nombre entré en argument
- La taille de noyau de convolution, c'est le second argument
- La fonction d'activation utilisée, nous utilisons ici la fonction unité linéaire rectifiée (ReLU)
- Le paramètre padding='same' signifie simplement que nous ne modifions pas la taille de l'image.

On a ensuite la couche « d'aplatissement » avec layers.Flatten() suivie de la partie classification constitué de couches entièrement connectées.

Les deux fonctions restantes sont Data_augmentation et layers.Dropout(0.3) qui permettent de lutter de contre le surajustement. Lorsqu'il existe un petit nombre d'exemples d'apprentissage, le modèle apprend parfois des bruits ou des détails indésirables des exemples d'apprentissage, à tel point que cela a un impact négatif sur les performances du modèle sur les nouveaux exemples. Ce phénomène est connu sous le nom de surajustement. Cela signifie que le modèle aura du mal à généraliser sur un nouvel ensemble de données.

Data_augmentation est une fonction permettant l'augmentation des données. L'augmentation des données consiste à générer des données d'entrainement supplémentaires à partir de nos exemples existants en les augmentant à l'aide de transformations aléatoires qui produisent des images d'apparence crédible.

```
data_augmentation = tf.keras.Sequential(
  [
    layers.RandomFlip("horizontal", input_shape=(img_width, img_height, 3)),
    layers.RandomFlip("vertical"),
  ]
)
```

Les transformations réalisées ici sont des retournements selon un axe horizontal et vertical.

La couche Dropout est une couche de suppression qui fonctionne en éliminant de manière aléatoire certaines des connexions entres les couches. 0,3 signifie qu'elle supprime 30% des connexions existantes.

II.2.2.3.3. Configuration et entrainement du réseau

Maintenant que notre modèle est créé, nous devons le compiler puis l'entrainer.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

epochs = 15
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

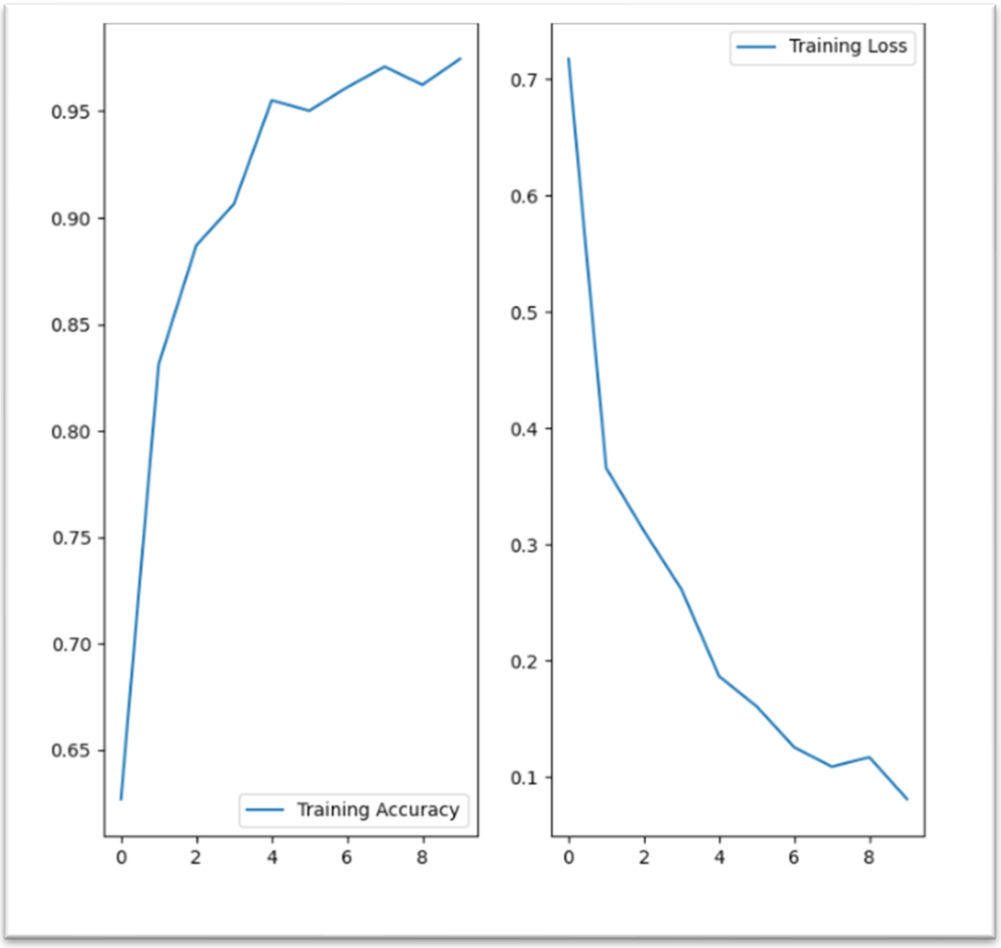
La compilation du modèle consiste à configurer le modèle pour la formation. Pour cela nous entrons en argument de la fonction compile les paramètres suivants :

- optimizer='adam' : l'optimiseur est ce qui réglera les pondérations de votre réseau pour approcher le point de perte la plus faible. L'optimisation Adam est une méthode de descente de gradient qui repose sur les moyennes courantes des gradients et des seconds moments des gradients.
- Une fonction de perte (ou de coût) : une fonction de perte, ou Loss function, est une fonction qui évalue l'écart entre les prédictions réalisées par le réseau de neurones et les valeurs réelles des observations utilisées pendant l'apprentissage. Dans notre cas, la fonction SparseCategoricalCrossentropy calcule la perte d'entropie croisée ou perte logarithmique, c'est à dire les performances d'un modèle de classification dont la sortie est une valeur de probabilité comprise entre 0 et 1.
- Une liste des paramètres à évaluer par le modèle lors de la formation et des tests, ici nous étudierons simplement la précision du modèle.

Notre modèle possède maintenant toutes les configurations nécessaires et nous pouvons donc passer à l'entrainement. Nous appelons pour cela la méthode fit, dans laquelle nous rentrons en paramètre le dataset d'entrainement, celui de validation ainsi que le nombre d'époque, c'est-à-dire le nombre de boucles d'entrainement.

II.2.2.3.4. Graphique et tests

Nous pouvons facilement visualiser l'évolution de la précision de notre réseau :



Nous pouvons également calculer la précision de notre modèle sur le dataset de validation :

```
val ds, accuracy: 96.59%
```

Et finalement, nous pouvons réaliser des tests avec différentes photos :

```
Cette image représente surement un(e) Masque
avec une probabilité de 100.00%.
```


```
Cette image représente surement un(e) boite
avec une probabilité de 93.97%.
```

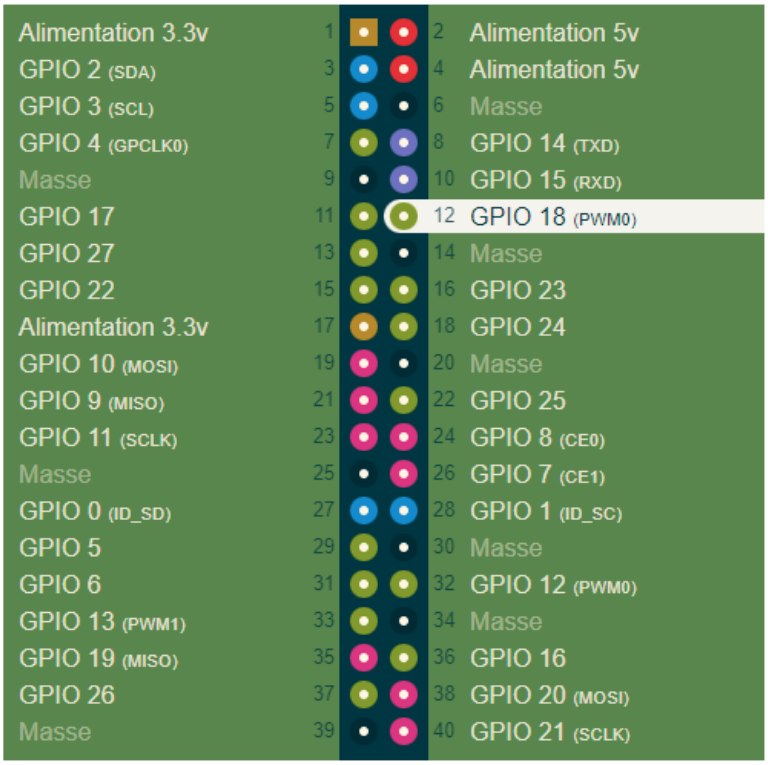
```
Cette image représente surement un(e) gobelet
avec une probabilité de 96.38%.
```

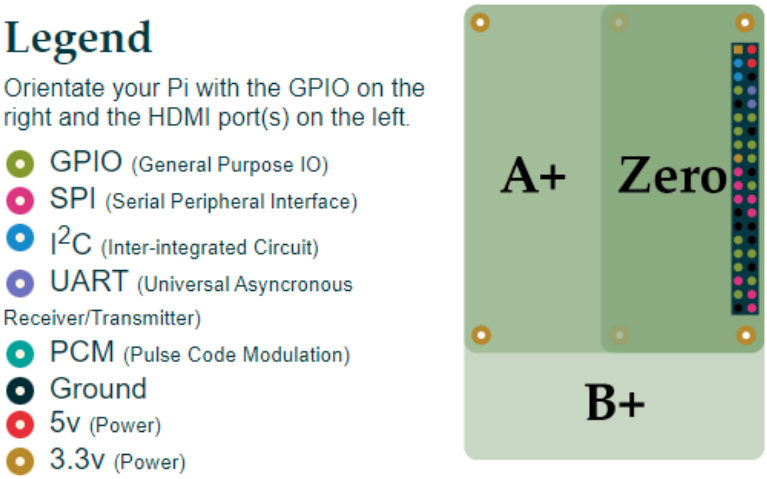
Notre réseau peut désormais reconnaitre les déchets et il ne nous reste plus qu’à les trier.

III. Le tri effectif des déchets

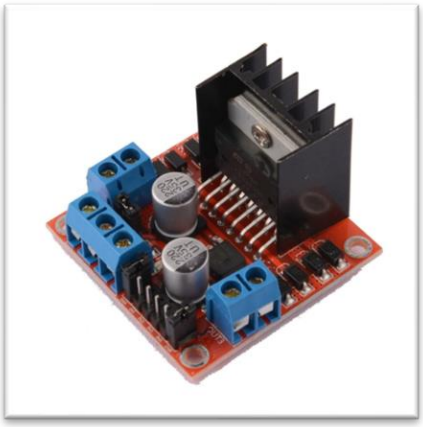
III.1. La connexion des différents composants au Raspberry Pi

Comme expliqué précédemment, le Raspberry Pi possède 40 broches GPIO, celles-ci ont beaucoup de propriétés uniques. Ainsi, lorsque l'on connecte notre composant à celles-ci, on ne peut pas utiliser n'importe qu'elle broche. Il existe 8 types de broches GPIO : les classiques, les SPI, les I^2C, les UART, les PCM, les masses, les alimentations de 5V et l'alimentation de 3,3V.



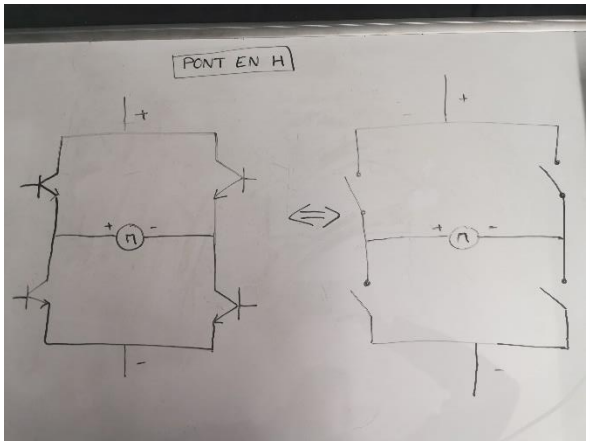


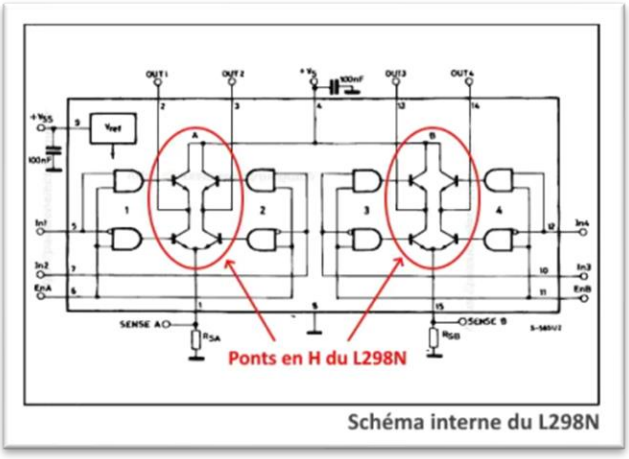
Tout d'abord, nous allons connecter notre moteur au Raspberry Pi, pour cela nous sommes passés par un hacheur, plus précisément le L298N.



En effet, ce module permet de contrôler jusqu'à deux moteurs à courants continus et nous offre la possibilité de choisir, entre, faire tourner notre moteur en vitesse continue ou en PWM. Cela signifie que nous pouvons choisir à la fois le sens de rotation et la vitesse de celui-ci.

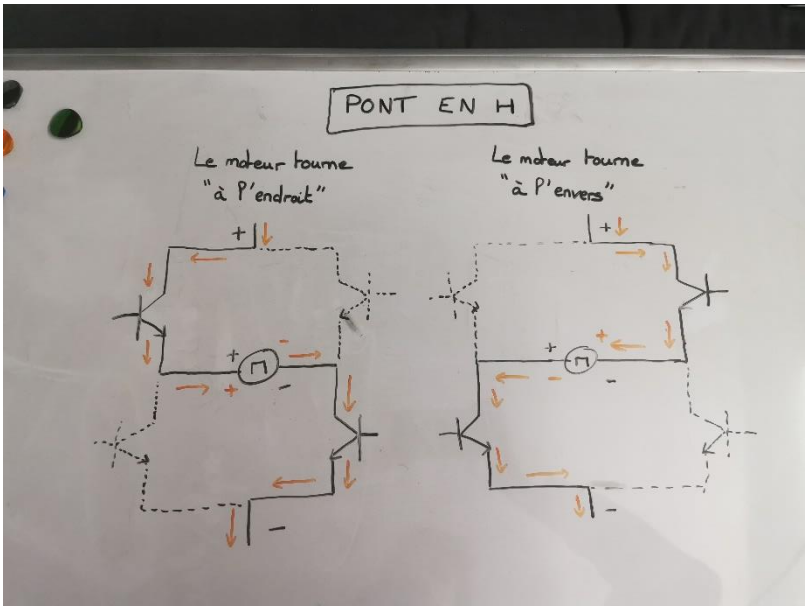
Pour contrôler le sens de rotation du moteur le principe est simple, notre hacheur est un double pont en H. On lui donne ce nom car les composants électroniques sont placés de manière particulière. Il est en général composé de quatre transistors, disposés d'une certaine manière, formant visuellement la lettre « H ».





En activant les transistors deux par deux (ceux de sens opposés), on peut contrôler le sens du passage du courant dans la charge, branchée « au milieu du H ». Et c'est ce changement de sens de courant, qui permet aux moteurs à courant continu de pouvoir tourner « à l'endroit », ou « à l'envers ».

Nous pouvons le représenter schématiquement :



Dans le cas de gauche : le « + » du moteur est sur le « + » de l'alimentation, et le « – » du moteur est sur le « – » de l'alimentation. Ainsi, le moteur tourne « à l'endroit »

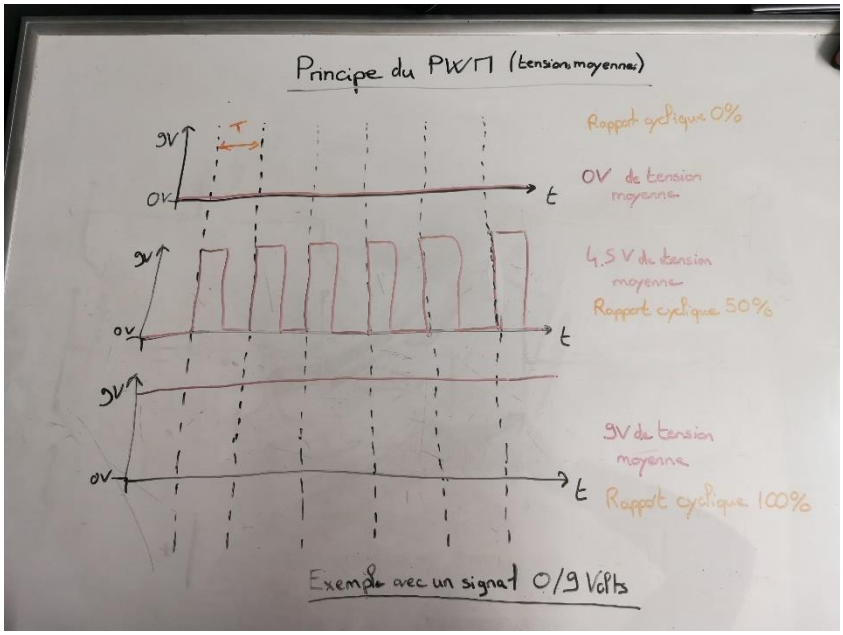
A l'inverse, dans le cas de droite : le « + » du moteur est sur le « – » de l'alimentation, et le « – » du moteur est sur le « + » de l'alimentation. Ainsi, le moteur tourne « à l'envers »

Pour résumer, il nous suffit de commander deux des quatre transistors de manière symétrique. Pour facilement faire varier le sens du courant et donc le sens du moteur.

Pour contrôler la vitesse de rotation du moteur, nous utilisons le fait que notre hacheur peut être piloté via un signal à rapport cyclique variable (PWM)

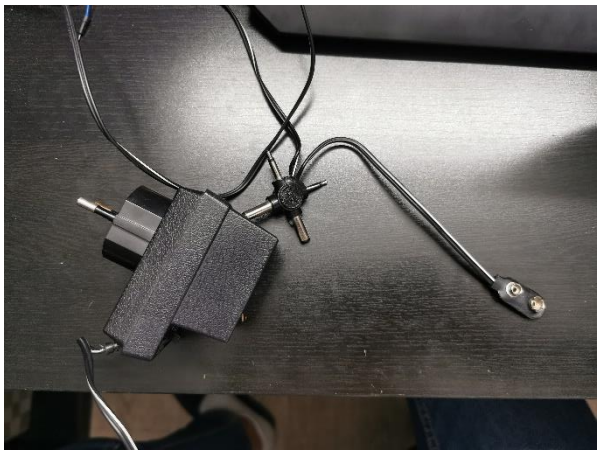
Le principe est tout aussi simple : la vitesse d'un moteur à courant continu est proportionnelle à sa tension d'alimentation autrement dit sa tension moyenne. Ainsi, il nous suffit de faire varier la tension moyenne de l'alimentation fournie à un moteur, pour en faire varier sa vitesse de rotation. Et pour faire varier une tension moyenne, il nous faut moduler la largeur de l'impulsion.

En effet, en PWM, on génère un signal de fréquence fixe, et de rapport cyclique variable. Et mathématiquement, la tension moyenne sera tout simplement égale à la tension max, multipliée par ce rapport cyclique.



Ainsi, pour faire varier la vitesse de notre moteur, il nous suffit d'envoyer un signal PWM au module. Chaque « pont en H » hachera alors la tension d'alimentation des moteurs à ce rythme, et la vitesse de rotation des moteurs sera parfaitement contrôlée.

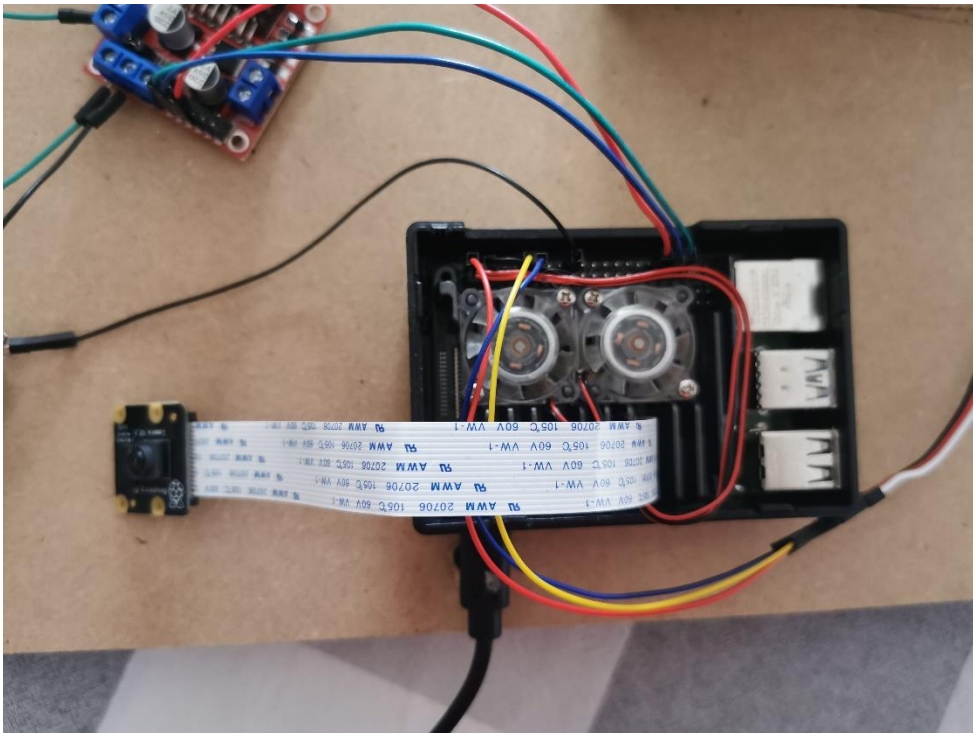
Il ne nous reste plus qu'à relier l'alimentation de notre moteur, sur les deux ports situés sur la droite ou sur la gauche du module et d'alimenter correctement le module. Pour alimenter le module, nous utilisions une pile 9V mais nous nous sommes vites rendu compte que cela n'était pas viable car après seulement quelques semaines de travail (la pile n'est branché que lors de nos tests sur le tapis roulant) nous avions vidé plusieurs piles. Ainsi, nous sommes passé sur un adaptateur d'alimentions dont nous pouvions moduler la tension d'entrée entre 3V et 12 V par bond de 1.5V. Pour finir, on connecte le module au Raspberry Pi en utilisant les broches GPIO numéro 36, 38 et 40.



Comme nous le disions précédemment, nous avons aussi connecté un ventilateur à notre Raspberry Pi pour éviter qu'il ne chauffe lorsqu'il effectue la détection et l'analyse des différents objets qui lui sont proposés. Pour cela, nous avons utilisé la première broche GPIO c'est-à-dire l'alimentation de 3.3 Volts.

Pour finir, nous avons connecté le servomoteur en utilisant les broches : GPIO 2 GPIO 6 et GPIO 12.

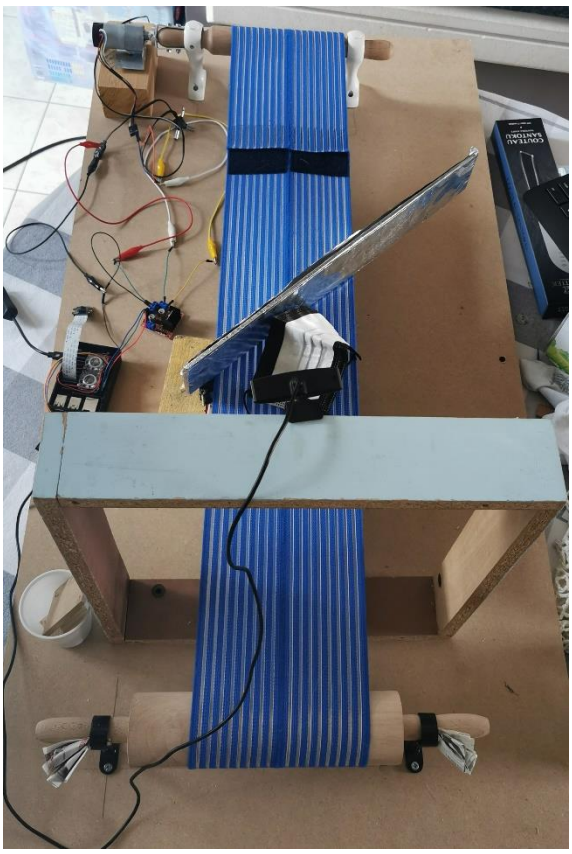
La deuxième broche GPIO correspond à l'alimentation de 5 Volts, la sixième correspond à la masse et la douzième est celle qui permet de transmettre le signal PWM.



III.2. Le servomoteur

Pour trier les différents déchets nous nous sommes inspirés d'une de nos premières idées c'est-à-dire celle du train. Nous avons retenu dans celle-ci, le principe d'aiguillage et nous avons donc décidé de l'adapté à notre tapis. Pour cela, nous avons équipé notre servomoteur d'une baguette de bois sur laquelle nous avons fixé une petite plaque de bois. Cette plaque a été rajoutée pour éviter que des déchets ne passent en dessous ou au-dessus de la baguette de bois.

C'est grâce à cette plaque que le déchet peut être trié. En effet, lorsque le déchet passe sous la caméra, l'information est traitée par le Raspberry Pi. Celle-ci donne alors un ordre au servomoteur, qui se déplace de manière à prendre l'angle qui lui est imposé. Ainsi la plaque de bois vient dévier le déchet de sa trajectoire initiale pour le guider vers la poubelle qui lui est attribué.



Photos du montage global avec le servomoteur à 15° et 30°

Conclusion

Notre système peut, en l'état actuel, trier trois types de déchets que sont les masques, les gobelets en plastique et les boîtes de Doliprane. Il est évident que l'on peut adapter notre système pour qu'il trie des déchets différents de ceux que nous avons choisis ou qu'il en trie tout simplement plus.

Une des améliorations possibles du système serait d'agrandir le tapis et d'ajouter un servomoteur de manière à effectuer un tri plus rapide. En effet, lorsque la caméra repère un déchet, il est nécessaire qu'aucun autre ne soit présent sur le tapis (le déchet précédent doit être tombé dans sa poubelle).

De plus, notre système ne possède aucun moyen de détecter le déchet en soit, il s'occupe simplement de classer l'image que lui transmet la caméra dans une des classes disponibles. Ainsi, un des axes majeurs d'amélioration serait d'ajouter, en plus de la classification des déchets, leur détection. En effet, cela éviterait au réseau d'être sollicité en permanence ce qui permettrait de limiter la consommation d'énergie du Raspberry Pi et d'améliorer ses performances globales (efficacité, longévité).

Bibliographie

• BERGER Laurent, *Traitement d'images et de vidéos avec avec OpenCV 4,* D-Booker, 2020

• KARVINEN Tero, KARVINEN Kimmo, Valtokari Ville, *Les capteurs pour Arduino et Raspberry Pi – Tutoriels et projets*, Dunod, 2014

• CLEMENT Patrice, *Python et Raspberry Pi – Apprenez à développer sur votre nano-ordinateur (3e édition)*, ENI, 2021

• HALFACREE Gareth, *Le Guide Officiel du débutant Raspberry Pi – Comment utiliser votre nouvel ordinateur (4e édition) (196-213)*, Raspberry Pi Press ,2020

Sitographie

• *Premiers pas avec la reconnaissance d'image à l'aide de TensorFlow et de Raspberry Pi* [en ligne], [consulté le 10 octobre 2021]. *Ephesos Software*. Adresse de l'article : https://fr.ephesossoftware.com/articles/diy/get-started-with-image-recognition-using-tensorflow-and-raspberry-pi.html

• KUENTZ Stéphane, *Framboise 314* [en ligne], 30 septembre 2020 [consulté le 4 novembre 2021]. *Vision artificielle : Testez la technologie TensorFlow / OpenCV4 sur Raspberry Pi 4*. Adresse de l'article : https://www.framboise314.fr/vision-artificielle-testez-la-technologie-tensorflow-opencv4-sur-raspberry-pi-4/

• KUENTZ Stéphane, *Framboise 314* [en ligne], 17 janvier 2019 [consulté le 4 novembre 2021]. *I.A : Créez votre propre modèle de reconnaissance d'objets (1er partie).* Adresse de l'article : https://www.framboise314.fr/i-a-creez-votre-propre-modele-de-reconnaissance-dobjets-1er-partie/

• *Raspberry Pi FR* [en ligne], 14 janvier 2020 [consulté le 20 novembre 2021]. *Comment contrôler un servomoteur avec la Raspberry Pi.* Adresse de l'article : https://raspberry-pi.fr/servomoteur-raspberry-pi/

• Pysource*, YouTube* [en ligne], 19 mai 2020 [consulté le 1 décembre 2021*]. Identify objects moving on a conveyor belt using OpenCV with Python*. Adresse de l'article : https://www.youtube.com/watch?v=A29IqeahI84

• Chronophage, YouTube [en ligne], [consulté le 15 décembre 2021]. Le fonctionnement des réseaux de neurones – Introduction au Deep Learning. Adresse de la page : https://www.youtube.com/watch?v=T-wpJPZV8io

• Dan Nelson, StackAbuse [en ligne], 13 avril 2022 [consulté le 20 décembre 2021]. Image Recognition and Classification in Python with TensorFlow and Keras. Adresse de la page : https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/

• TensorFlow [en ligne], 26 janvier 2022 [consulté le 21 décembre 2021]. Image classification. Adresse de la page : https://www.tensorflow.org/tutorials/images/classification

The Data Frog [en ligne], [consulté le 21 décembre 2021]. Entre chien et chat. Adresse de la page : https://thedatafrog.com/fr/articles/dogs-vs-cats/

• TensorFlow [en ligne], 30 avril 2022 [consulté le 21 décembre 2021]. Module : tf. Adresse de la page : https://www.tensorflow.org/api_docs/python/tf

• Gary B, DataScienctest [en ligne], 25 juin 2020 [consulté le 8 janvier 2022]. Convolutional neural network. Adresse de la page : https://datascientest.com/convolutional-neural-network

• Defend Intelligence, YouTube [en ligne], [consulté le 15 janvier 2022]. Coder un réseau de neurones convolutifs de classification d'image avec Python et Tensorflow.
Adresse de la page : https://www.youtube.com/watch?v=6FHtTyZxS5s

• Julien Krywyk, Pierre-Alain Jachiet, Le Blog des Octos [en ligne], 25 octobre 2016 [consulté le 17 janvier 2022]. Classification d'image : les réseaux de neurones convolutifs en toure simplicité.
Adresse de la page : https://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/

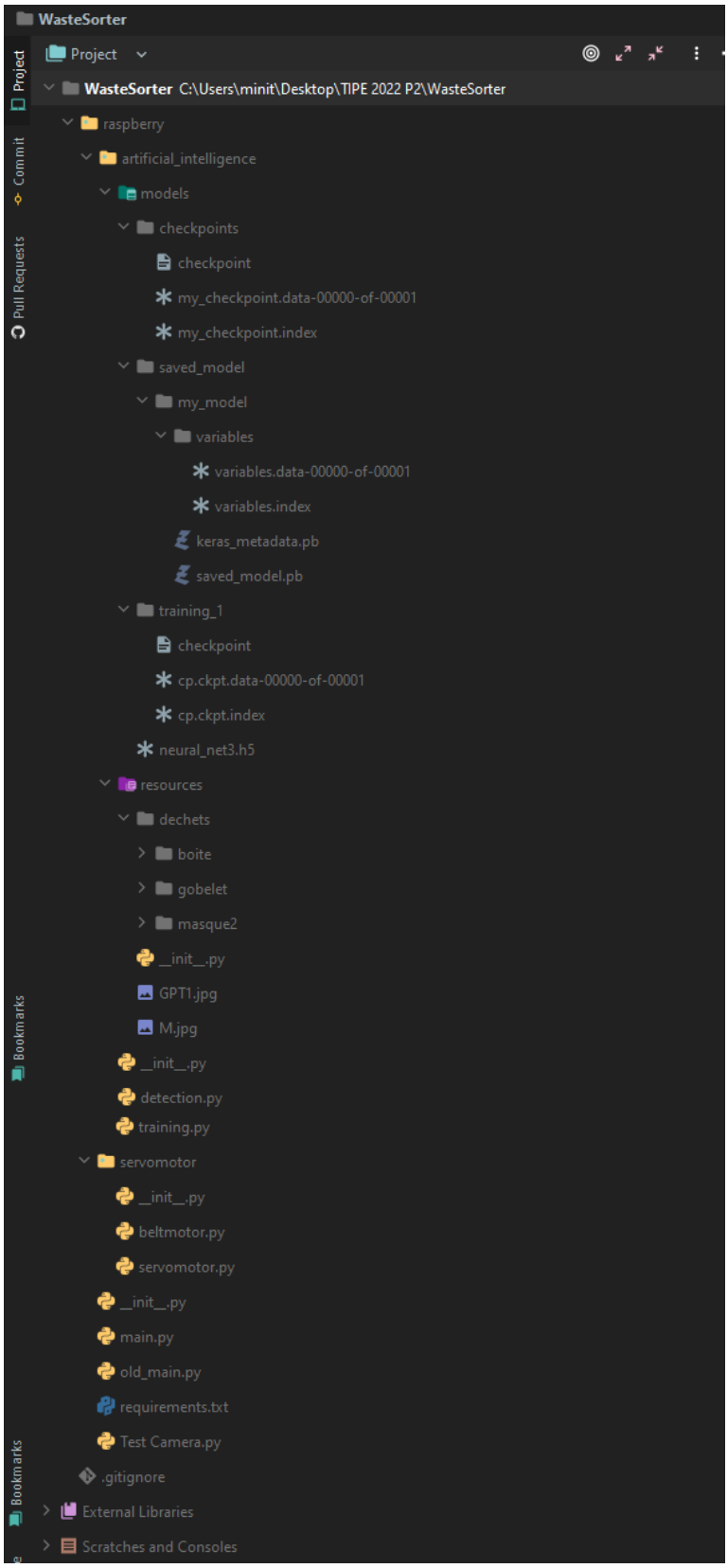
• DFROBOT [en ligne], [consulté le 28 février 2022]. Gear Motor w/Encoder model No.GB37Y3530-12V-251R. Adresse de la page :
https://wiki.dfrobot.com/12V_DC_Motor_251rpm_w_Encoder__SKU__FIT0186_

• CFAURY, arduino.blaisepascal [en ligne], 22 février 2021 [consulté le 1 mars]. Pont en H L298N.
Adresse de la page : https://arduino.blaisepascal.fr/pont-en-h-l298n/

• natural solutions [en ligne], 17 avril 2018 [consulté le 23 mars 2022]. Les réseaux de neurones convolutifs. Adresse de la page : https://www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones-convolutifs#:~:text=Qu'est%2Dce%20que%20la,%C3%A0%20la%20reconnaissance%20d'image.

Annexe

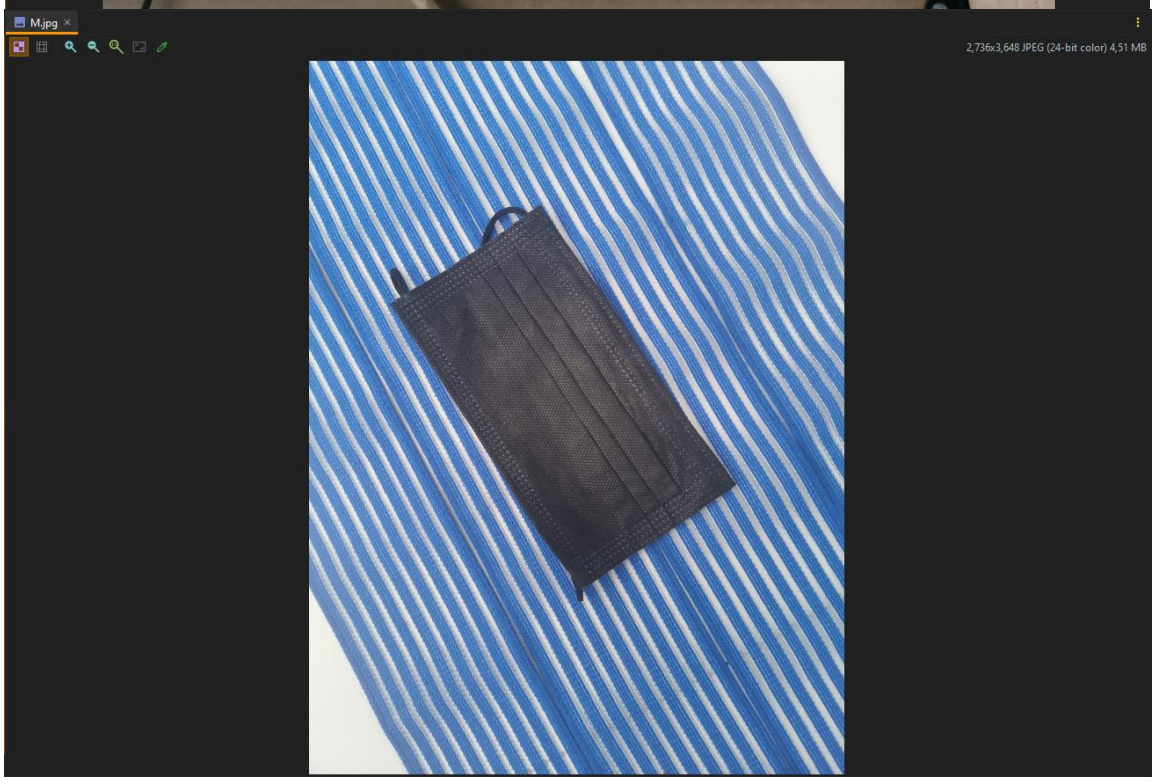
Voici la structure de notre projet avec tous les codes que nous avons utilisés :



Les dossiers : « boite », « gobelet » et « masque2 » contiennent les photos d'entrainement du réseau de neurone.

checkpoint

```
model_checkpoint_path: "my_checkpoint"
all_model_checkpoint_paths: "my_checkpoint"
```



detection.py

```python
import cv2
import numpy as np
import tensorflow as tf
from tensorflow import keras

img_height = 180
img_width = 180
batch_size = 32

train_ds = tf.keras.utils.image_dataset_from_directory('resources/dechets/', validation_split=0.2, subset="training",
                                                       seed=123, image_size=(img_height, img_width),
                                                       batch_size=batch_size)

class_names = train_ds.class_names
model = keras.models.load_model('models/neural_net3.h5')

largeur_image = 180
hauteur_image = 180


def detect(img):
    # On récupère les données de la photo
    imgResized = cv2.resize(img, (largeur_image, hauteur_image))
    imgArray = tf.keras.utils.img_to_array(imgResized)
    imgArray = tf.expand_dims(imgArray, 0)  # problème de dimension (480, 640)

    predictions = model.predict(imgArray)
    score = tf.nn.softmax(predictions[0])

    return class_names[np.argmax(score)], 100 * np.max(score)

```

training.py

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

#
img_height = 180
img_width = 180
batch_size = 32

train_ds = tf.keras.utils.image_dataset_from_directory('resources/dechets/', validation_split=0.2, subset="training",
                                                       seed=123, image_size=(img_height, img_width),
                                                       batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory('resources/dechets/', validation_split=0.2, subset="validation",
                                                     seed=123,
                                                     image_size=(img_height, img_width), batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)

AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
num_classes = len(class_names)

data_augmentation = tf.keras.Sequential(
    [
        layers.RandomFlip("horizontal", input_shape=(img_width, img_height, 3)),
        layers.RandomFlip("vertical"),
    ]
)
model = Sequential([
    data_augmentation,
    layers.Rescaling(1. / 255),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),

    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),

    layers.Conv2D(128, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),

    layers.Conv2D(256, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),

    layers.Flatten(),
    layers.Dense(128),
    layers.Dropout(0.4),
    layers.Dense(num_classes)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
epochs = 15
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

# Save the entire model as a SavedModel.
model.save('models/neural_net3.h5')
loss2, acc2 = model.evaluate(val_ds, verbose=2)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc2))
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
# plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
# plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

test = tf.keras.utils.load_img('resources/masque_noir_test.png', target_size=(img_height, img_width))

plt.imshow(test)
plt.show()

test_array = tf.keras.utils.img_to_array(test)
test_array = tf.expand_dims(test_array, 0)  # Create a batch

predictions = model.predict(test_array)
print("prediction du test", predictions)
score = tf.nn.softmax(predictions[0])
print("prediction ", predictions)
print("score", score)
```

```python
if 100 * np.max(score) ⩽ 75:
    print("déchet non reconnu")
else:
    print("This image n1 most likely belongs to {} with a {:.2f} percent confidence.".format(
        class_names[np.argmax(score)],
        100 * np.max(score)))

test2 = tf.keras.utils.load_img('resources/GPT1.jpg', target_size=(img_height, img_width))
plt.imshow(test2)
plt.show()

test_array2 = tf.keras.utils.img_to_array(test2)
test_array2 = tf.expand_dims(test_array2, 0)  # Create a batch

predictions2 = model.predict(test_array2)
print("prediction du test 2", predictions2)
score2 = tf.nn.softmax(predictions2[0])
print("prediction 2", predictions2)
print("score 2", score2)

if 100 * np.max(score2) ⩽ 75:
    print("déchet non reconnu")
else:
    print("This image n2 most likely belongs to {} with a {:.2f} percent confidence.".format(
        class_names[np.argmax(score2)],
        100 * np.max(score2)))
```

beltmotor.py

```python
import RPi.GPIO as GPIO

# Definition des pins
M1_En = 21
M1_In1 = 20
M1_In2 = 16

# Creation d'une liste des pins pour chaque moteur pour compacter la suite du code
Pins = [[M1_En, M1_In1, M1_In2]]

# Setup
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(M1_En, GPIO.OUT)
GPIO.setup(M1_In1, GPIO.OUT)
GPIO.setup(M1_In2, GPIO.OUT)


# Initialisation

def start():
    M1_Vitesse = GPIO.PWM(M1_En, 100)

    M1_Vitesse.start(100)


def sens1(moteurNum):
    GPIO.output(Pins[moteurNum - 1][1], GPIO.HIGH)
    GPIO.output(Pins[moteurNum - 1][2], GPIO.LOW)


def sens2(moteurNum):
    GPIO.output(Pins[moteurNum - 1][1], GPIO.LOW)
    GPIO.output(Pins[moteurNum - 1][2], GPIO.HIGH)


def pause(moteurNum):
    GPIO.output(Pins[moteurNum - 1][1], GPIO.LOW)
    GPIO.output(Pins[moteurNum - 1][2], GPIO.LOW)
    print("Moteur", moteurNum, "arrêt.")


def stop():
    GPIO.output(Pins[0][1], GPIO.LOW)
    GPIO.output(Pins[0][2], GPIO.LOW)
    print("Moteurs arrêtés.")

```

```
49      # stop()
50      #
51      # print("Moteur 1 tourne dans le sens 1.")
52      # sens1(1)
53      # sleep(3)
54      # stop()
55      # sleep(3)
56      # print("Moteur 1 tourne dans le sens 2.")
57      # sens2(1)
58      # sleep(2)
59      # pause(1)
60      # sleep(1)
61      #
62      # print("Moteur 1 tourne dans le sens 1.")
63      # while True:
64      #     sens1(1)
65
```

servomotor.py

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

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)  # Use Board numerotation mode
GPIO.setwarnings(False)  # Disable warnings

# Use pin 12 for PWM signal
pwm_gpio = 12
frequency = 50
GPIO.setup(pwm_gpio, GPIO.OUT)
pwm = GPIO.PWM(pwm_gpio, frequency)


# Set function to calculate percent from angle
def angle_to_percent(angle):
    if angle > 180 or angle < 0:
        return False

    start = 4
    end = 12.5
    ratio = (end - start) / 180  # Calcul ratio from angle to percent

    angle_as_percent = angle * ratio

    return start + angle_as_percent


pwm.start(angle_to_percent(90))

def setBasePosition():
    """
    Set the base position of the servo
    """
    pwm.ChangeDutyCycle(angle_to_percent(90))


def rotate(angle):
    """
    Rotate the servo to the given angle
    :param angle: angle to rotate to
    :return: None
    """

    percent = angle_to_percent(angle)
    pwm.ChangeDutyCycle(percent)


def stop():
    """
    Stop the servo
    :return: None
    """
    pwm.stop()
    GPIO.cleanup()
```

```
59
60  object_to_angle = {
61      'masque': 90,
62      'boite': 75,
63      'gobelet': 60
64  }
65  #
66  # # Init at 0°
67  # pwm.start(angle_to_percent(90))
68  # time.sleep(1)
69  #
70  # # Go at 25°
71  # pwm.ChangeDutyCycle(angle_to_percent(75))
72  # time.sleep(10)
73  #
74  # # Go at 35°
75  # pwm.ChangeDutyCycle(angle_to_percent(60))
76  # time.sleep(10)
77  #
78  # # Return 0°
79  # pwm.ChangeDutyCycle(angle_to_percent(90))
80  # time.sleep(1)
81  #
82  # # Close GPIO & cleanup
83  # pwm.stop()
84  # GPIO.cleanup()
85
```

main.py

```python
import time

import cv2

import raspberry.servomotor.beltmotor as beltmotor
import raspberry.servomotor.servomotor as servomotor
from raspberry.artificial_intelligence.detection import detect

cam = cv2.VideoCapture(0)


def start():
    print("Démarrage du programme WasteSorter !")
    servomotor.start()
    time.sleep(2)
    beltmotor.start()

    while True:
        ret, image = cam.read()

        if not ret:
            continue

        classname, confidence = detect(image)
        print("Détection d'un", classname, "avec une confiance de", confidence)

        if classname != "boite" and confidence > 0.6:
            definedAngle = servomotor.object_to_angle[classname]
            if definedAngle:
                servomotor.rotate(definedAngle)

        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    servomotor.stop()
    beltmotor.stop()


if __name__ == '__main__':
    start()
```

Test Camera.py

```python
import cv2

# On accède à la webcam
cam = cv2.VideoCapture(0)

# On prend une photo
ret, image = cam.read()

# Paramètre pour définir la fenêtre
nom_fenetre = "video_cam"

largeur_image = int(cam.get(cv2.CAP_PROP_FRAME_WIDTH))
hauteur_image = int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))

cv2.namedWindow(nom_fenetre, cv2.WND_PROP_FULLSCREEN)

# Tout le temps
while True:
    # On prend une photo, ahh c'est pas en temps réel ?
    ret, image = cam.read()
    if ret:

        # On affiche la fenêtre avec notre image
        cv2.imshow(nom_fenetre, image)

        # On ajoute un temps d'attente
        # et on arrête la boucle si on appuie sur la touche 'q'
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
```

requirements.txt

```
matplotlib~=3.5.1
numpy~=1.22.2
Pillow==9.1.0
```