Plan rapport :

1. Présenter le projet (objectif, spécifications)
2. Concepts / / justification des choix (audio : langage naturel – tps imparti -> approche simple : un ordre= un mot) /Pipeline (dataset – 2 réseaux différents = 2 problèmes de classification) / choix de démo.
3. Le projet

* Librosa package : différentes représentations d’un .wav (ex : …)
* Réseaux de neurones : neurone formel / architecture (dense et convolution) - illustration KERAS code/ entrainement et validation (overfitting précautions)
* Robot UR

1. Résultats Train/Validation :

* Approche 1
* Approche 2

1. Implémentation : Real time analysis

* Interface Tkinter
* Problèmes rencontrées
* Résultats obtenues

1. Améliorations :

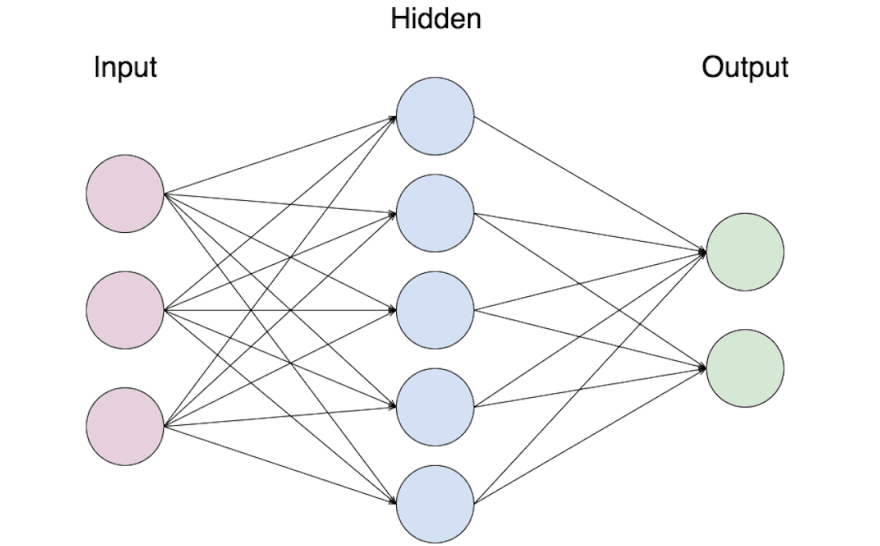
* RN avec KNN en dernière couche
* Etude de robustesse : Création des données en Atelier
* Avec ou sans normalisation/standardisation

1. **Présentation du projet : Objectifs, spécifications**

Dans un contexte d'usine de future, les opérateurs et les robots doivent travailler avec plus en plus en collaboration. C'est l'un des piliers de l'industrie de future. Afin de rendre ce but possible, il est nécessaire de travailler sur des nouvelles méthodes de communication entre ces deux entités, avec la façon la plus naturelle possible. C'est ainsi que notre projet d'expertise est créée afin de s'approcher de cette quête. Notre objectif est de pouvoir commander les robots collaboratifs avec la voix, car c'est le moyen le plus naturel de communication pour l'homme, et rendre cette tache possible dans un environnement aussi dure qu'une usine de production. Notre projet consiste à commander le robot afin qu'il puisse effectuer des taches simples, à savoir donner un outil à l'opérateur, sous des commandes simples constituées d'un seul mot. Ce robot devra pouvoir "comprendre" cette commande et discerner l'opérateur, afin de s'adapter à certaines exigences de celui-ci à savoir sa taille, sa main dominante et probablement son accès aux fonctionnalités du robot. Cette analyse devra être faite avec un réseau de neurones, pré entrainé avec des données pertinentes. Nous procéderons après à une démonstration des travaux dans un contexte d'utilisation de ce système pour l'assemblage d'une pièce mécanique.

1. **Concept :**
2. **Justification des choix :**

Notre problème consiste à discerner des mots ainsi que des opérateurs basé sur un répertoire interne de données. Nous avons choisie pour ce système une architecture basée sur un réseau de neurones du fait de son simplicité de conception et ses points forts en classification. En effet, les réseaux de neurones présentent l'avantage de nous épargner le traitement et l'analyse fastidieuse du signal audio afin de faire sortir des « règles » qui peuvent nous servir pour classifier les ordres. Ces réseaux de neurones nécessitent des données labélisés afin de les entrainer et les utiliser pour classifier les ordres.

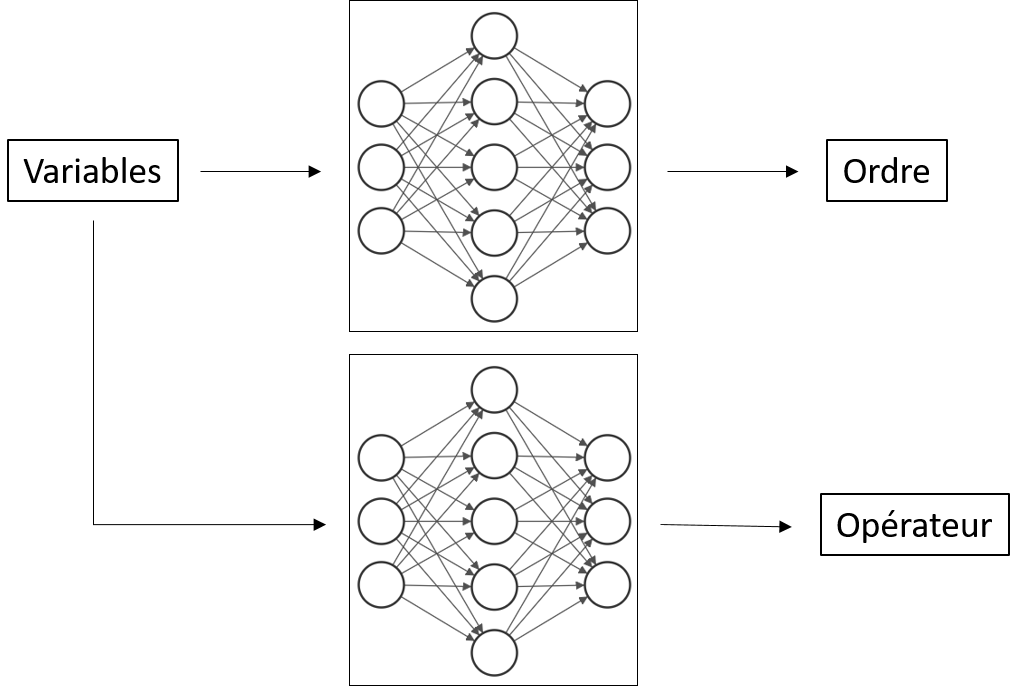


*Architecture classique d’un réseau de neurones Dense*

Comme c’est montré sur la figure, le nombre d’entrées d’un réseau de neurone complétement connecté (Dense) est fixe. C’est pourquoi il a fallu que les ordres qu’on prononce ne dépassent pas une durée totale d’une seconde (Ce choix nous été imposé par les données d’entrainement existantes sur internet). Ceci nous permet, pour n’importe quel pré-traitement du signal, d’avoir un nombre de variable fixe pour chaque cas d’ordre.

1. **Pipeline de l’architecture :**

Notre système est composé de 2 réseaux de neurones qui acceptent les mêmes variables calculées sur le signal audio après préprocessing. Le premier est celui de classification d’ordre, et l’autre pour la classification d’opérateur. Il est important de noter que nous avons travaillé avec 2 approches différentes pour extraire les variables à partie du signal. Cette étape va être expliquée dans une partie dédiée.

**

1. **Choix de la Démo :**

Pour démontrer nos travaux dans un contexte industriel qui pourra démontrer le système en action, nous avons proposé de mettre en œuvre un assemblage d’un vérin, composé de 2 pièces, avec l’assistance du robot qui va s’adapter à chaque fois à un des 2 opérateurs qui vont travailler avec.

1. **Le projet**

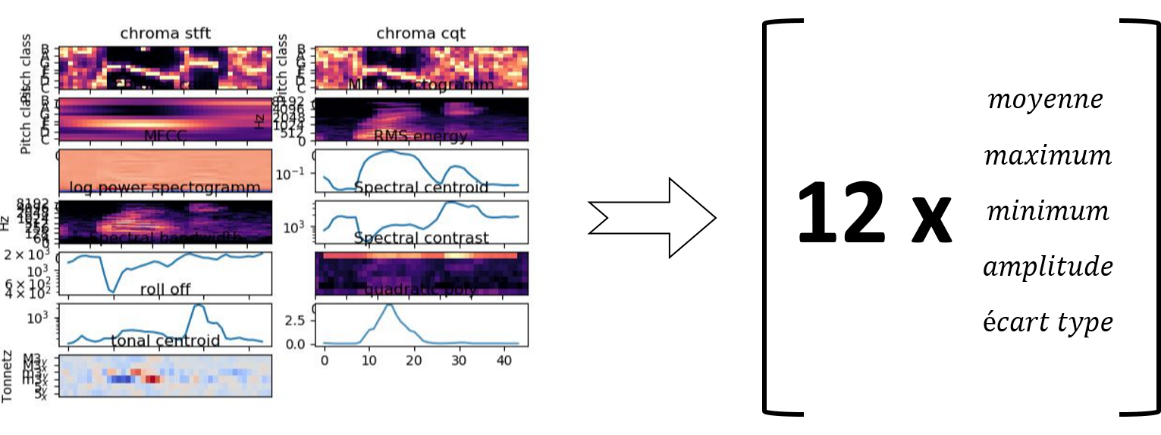
Afin d’expliquer le fonctionnement du système, nous allons choisir le même flow que celui des données dans le pipeline. Nous allons commencer par le pré-traitement du signal audio par Librosa, ensuite on va passer au réseau de neurones, afin de les expliquer d’une façon succincte et présenter le Framework et les librairies choisies, les principales notions statiques et quelques précautions lors de l’utilisation de ces architectures, ensuite pour passer au robot qu’on va s’en servir pour tester et démontrer le système.

1. **La librairie Librosa :**
2. **Les réseaux de neurones :**
3. **Le robot « Universel Robot » UR :**

***Les features :***

Cette deuxième méthode consiste à utiliser les features générées par Librosa afin de calculer 5 variables statistiques à partir de chacun des 12 spectrogrammes choisis. Les spectrogrammes sont des matrices (types np.array float 32) qui contiennent les …… sous un format qui dépend de la longueur du fichier audio. L’approche 1 consistait à enregistrer les images de ces graphes, ces images sont en théorie de la même taille. Pourtant, en travaillant directement avec ces matrices, les RN auront à chaque fois un nombre d’entrées différend.

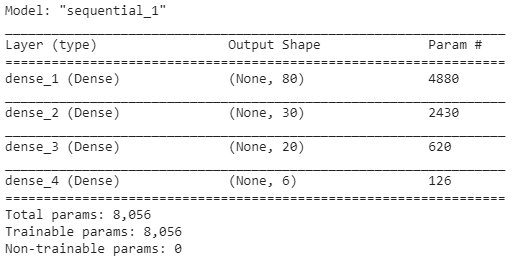
L’introduction de ces variables statistiques vise à pallier ce problème. A chaque fois, on calcul la moyenne, le minimum, le maximum, l’amplitude (max-min) et l’écart-type. Cette approche nous laisse avec 60 variables (5x12 matrices).



Cette approche se base sur le package extacrt\_features.py, créée durant ce projet. Le temps nécessaire afin de traiter un fichier audio d’une seconde est estimé à …. .

***Le réseau de neurones :***

Le réseau utilisé dans cette partie aura 60 entrées, ce qui fait qu’il est plus léger que celui de l’approche 1.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| optimizer | acuracy train | loss train | acuraty test | loss test |
| **SGD** |  |  |  |  |
| **RMSprop** |  |  |  |  |
| **Adagrad (epoch=10)** |  |  |  |  |
| **Adagrad (epoch=20)** |  |  |  |  |
| **Adadelta** |  |  |  |  |
| **Adam amsgrad=False** |  |  |  |  |
| **Adam amsgrad=True** |  |  |  |  |
| **Adamax** |  |  |  |  |
| **Nadam** |  |  |  |  |

La configuration retenue est :

Type : Sequential Dense

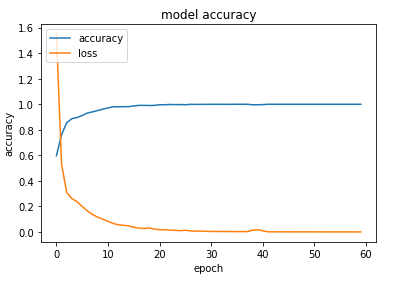
Fonctions d’activation : ReLu et softmax

Loss : categorical\_crossentropy

Optimizer : Adam

***Entrainement pour classification de commandes :***

Le graphe suivant montre l’évolution de la précision et de la perte (i.e. erreur) avec les époques. On peut voir que ce réseau de neurones peut être entrainé rapidement (convergence au bout de 30 époques).



***Entrainement pour classification de l’opérateur:***