

Alexandre Berard, Julien Liottard, Marie Vialle

Client : Jean-Pierre Ceysson, CARI Electronic

Tuteur : André Lagrèze

Projet de deuxième année de DUT Informatique

IUT de Valence

# Résumé des mêlées de la première phase de développement du projet IA Embarquée

26 janvier 2020 (version 5)

## Sommaire

<b>1</b>	<b>Cycle 1</b>	<b>2</b>
1.1	Installer Tensorflow . . . . .	2
1.2	Recherche de documentation sur l'intelligence artificielle . . . . .	3
1.2.1	Construire et comprendre un réseau de neurones virtuels . . . . .	3
1.2.2	Effectuer différents tests . . . . .	4
1.2.3	Réfléchir aux formats de données . . . . .	4
1.2.4	Explications théoriques et mathématiques sur les réseaux de neurones . . . . .	4
1.2.5	Gérer les chemins relatifs au système . . . . .	5
1.3	Tester les neurones avec des petites courbes . . . . .	5
1.4	Travail sur la carte . . . . .	9
1.4.1	Création d'un projet sous STM32CubeMX . . . . .	9
1.4.2	Configuration . . . . .	10
1.4.3	X-CUBE-AI . . . . .	11
<b>2</b>	<b>Cycle 2</b>	<b>12</b>
2.1	Acquisition de courbes . . . . .	12
2.2	Standardisation du format des données pour le Machine Learning . . . . .	12
2.3	Différenciation entre plusieurs cartes . . . . .	13
2.4	Adaptation du réseau sur la carte . . . . .	14
<b>3</b>	<b>Cycle 4</b>	<b>15</b>
3.1	Implémenter le réseau de neurones . . . . .	15
3.1.1	Prise en main de la carte . . . . .	15

# 1 Cycle 1

## 1.1 Installer Tensorflow

Afin d'installer TensorFlow, nous avons suivis des instructions que nous avons trouvé sur le site de tensorflow<sup>1</sup>, que nous allons redétailler ici-bas. Nous avons utilisé le terminal ubuntu linux sur windows.

Tout d'abord, nous avons vérifié que python et son environnement étaient bien configurés via les commandes :

```
python3 --version
pip3 --version
virtualenv --version
```

Si les packages ne sont pas bien installés, nous utilisons :

```
sudo apt update
sudo apt install python3-dev python3-pip
sudo pip3 install -U virtualenv
```

Nous avons ensuite créé un environnement virtuel :

```
virtualenv --system-site-packages -p python3 ./venv
```

Afin d'activer l'environnement, nous utilisons la commande :

```
source ./venv/bin/activate
```

Une fois que virtualenv est actif, le prompt du shell possède le préfixe (venv)

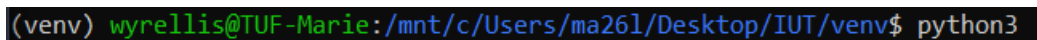
A screenshot of a terminal window with a black background. The prompt is shown in green text as (venv) wyrellis@TUF-Marie:/mnt/c/Users/ma261/Desktop/IUT/venv\$. The command python3 is entered in white text.

Figure 1 : Visuel du préfixe *venv* sur le prompt du shell

On continue en améliorant pip pour qu'il ait toutes les fonctionnalités à jour :

```
pip install --upgrade pip
pip list
```

Pour finir, afin de quitter l'environnement virtuel, nous utilisons :

```
deactivate
```

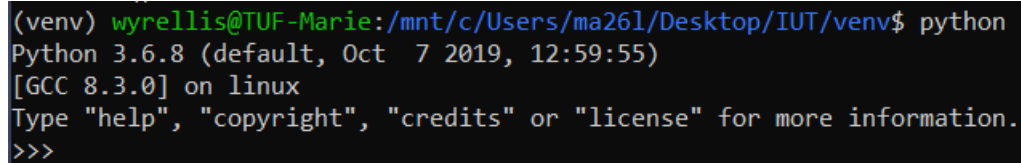
---

<sup>1</sup><https://www.tensorflow.org/install/pip?lang=python3>

Puis pour utiliser tensorflow, nous pouvons lancer python via

```
python3
```

qui nous permet d'ouvrir le shell python.



```
(venv) wyrellis@TUF-Marie:/mnt/c/Users/ma261/Desktop/IUT/venv$ python
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 2 : Visuel du shell python

## 1.2 Recherche de documentation sur l'intelligence artificielle

Nous avons durant ce premier sprint, utilisé de nombreuses documentations que nous allons lister ici, tout en donnant leur utilité sur notre projet.

### 1.2.1 Construire et comprendre un réseau de neurones virtuels

Tout d'abord, nous avons utilisé la documentation tensorflow<sup>2</sup> pour nous permettre de comprendre comment est construit un réseau de neurones simple, qui peut reconnaître des images, apprendre, et donner le taux de précision associé à ses estimations.

D'après les codes que nous avons pu voir, nous en avons déduis certaines informations :

```
model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test, verbose=2)
```

Ici, nous en avons déduit que "model" correspond à notre réseau de neurones, sur lequel nous pouvons appliquer deux méthodes : `.fit` et `.evaluate`. `.fit` permet de faire apprendre le réseau de neurones avec un dataset<sup>3</sup> (ici mnist via le lien de la partie 1.2.2) contenant les éléments à analyser (images ou vecteurs) et leur label correspondant.

---

<sup>2</sup><https://www.tensorflow.org/tutorials/quickstart/beginner>

<sup>3</sup>ensemble de données regroupées en masse, ici on utilise on regroupe 60000 images avec leurs labels



Figure 3 : Visuel des images et de leurs labels

.`evaluate` permet lui de tester le réseau de neurones. Il utilise un dataset comme le `.fit` mais pour donner une estimation de la précision du réseau de neurones qui a précédemment été calculé (grâce à l'apprentissage). Il est donc important de mettre des valeurs différentes pour le dataset de test et le dataset d'apprentissage.

### 1.2.2 Effectuer différents tests

Nous avons aussi testé le réseau de neurones que nous avons créé grâce à la documentation précédente avec d'autres banques d'images que nous avons trouvé sur internet<sup>4</sup>.

### 1.2.3 Réfléchir aux formats de données

Ensuite, pour fournir les courbes à python (en tant qu'entrées), nous devons utiliser un format CSV, nous avons donc cherché de la documentation sur le sujet<sup>5</sup>.

### 1.2.4 Explications théoriques et mathématiques sur les réseaux de neurones

Afin de comprendre de manière claire le fonctionnement d'un réseau de neurones au sens théorique et mathématiques, nous nous sommes intéressés à un livre<sup>6</sup> et nous avons regardé

<sup>4</sup><http://yann.lecun.com/exdb/mnist/>

<sup>5</sup><https://code.tutsplus.com/tutorials/how-to-read-and-write-csv-files-in-python--cms-29907>

<sup>6</sup>TensorFlow pour le deep learning, de la régression linéaire à l'apprentissage par renforcement - Bharath Ramsundar et Reza Bosagh Zadeh

deux vidéos sur youtube :

- Neural Networks from the ground up<sup>7</sup> (3Blue1Brown)
- How machines learn<sup>8</sup> (3Blue1Brown)

### 1.2.5 Gérer les chemins relatifs au système

Afin de gérer les chemins relatifs au système d'exploitation. Grâce à cette documentation<sup>9</sup>, on a compris comment récupérer le chemin du fichier où est le script python.

## 1.3 Tester les neurones avec des petites courbes

Tout d'abord, nous avons créé un script python qui permet de générer des courbes simples de 10 points.

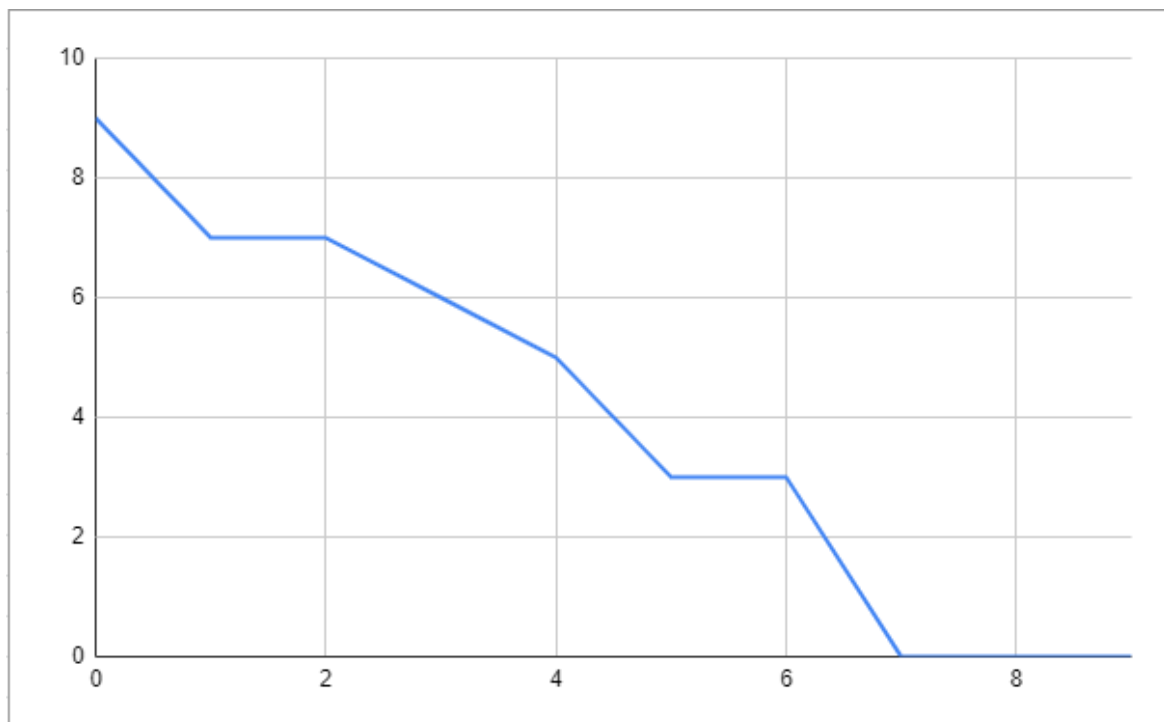


Figure 4 : Visuel d'une courbe

Nous en avons réalisé deux sortes : les courbes croissantes et décroissantes. Pour entraîner le réseau, nous avons fait quelques variations ce qui rend chaque courbe différente.

<sup>7</sup><https://www.youtube.com/watch?v=aircAruvnKk>

<sup>8</sup><https://www.youtube.com/watch?v=IHZwWFHwa-w>

<sup>9</sup><https://stackoverflow.com/questions/918154/relative-paths-in-python>

Nous avons décidé d'enregistrer ces graphiques dans des fichiers CSV, pour nous habituer à le faire lorsqu'il s'agira de récupérer des données sur l'oscilloscope, où ici nous n'aurons d'autre choix que cette extension. Pour ce faire, nous utilisons les fonctionnalités de Python qui permettent d'écrire et lire dans ces fichiers. La création de fichiers CSV se fait dans le fichier `tenpointsvectorgenerator.py`<sup>10</sup>, et la lecture de ceux-ci s'effectue dans `tenpointsvectorreader.py`<sup>11</sup>.

Nous devons associer à chaque courbe un label le décrivant, ici, dans cet exemple, nous distinguons une courbe croissance (ASC) d'une courbe décroissante (DESC), ce label sera inclus dans le fichier CSV. Avec ceci, nous pouvons utiliser un réseau de neurones permettant de distinguer une courbe qui croît d'une qui décroît.

Grâce aux ressources et exemples en ligne<sup>12</sup>, nous avons pu à minima comprendre quel est le format de données que peut prendre les réseaux de neurones de TensorFlow. TensorFlow peut apprendre les structures de données de NumPy<sup>13</sup>, nous devons donc convertir nos graphiques et labels en structure NumPy compréhensible par TensorFlow, ce dernier pouvant prendre en entrée une liste de valeurs entre 0 et 1 et un label pour l'apprentissage du réseau de neurones. Pour ce faire, les valeurs du graphique doivent être formatées (ce sont des nombres entiers sur un axe de nombre entiers de 0 à 10), on procède comme suit :

- On soustrait toutes les valeurs du graphique par son minimum, cela aura pour effet de mettre la valeur minimale du graphique à 0.
- On divise toutes ces valeurs par le nouveau maximum, ce qui positionnera toutes les valeurs du graphique dans l'intervalle  $[0,1]$ .

Cette procédure est effectuée dans la fonction `format_vector` du fichier `tenpointsvectorIA.py`<sup>14</sup>. Le label doit également être formaté, puisqu'on ne peut pas donner de variable de type string à notre réseau, mais que des types entiers. On choisit de mettre les labels dans un tableau<sup>15</sup> et les représenter sous forme d'entiers (on associera un label à son index dans le tableau). Ce formatage est fait dans la fonction `format_label` du même fichier (avec le label passé en paramètre).

---

<sup>10</sup><https://github.com/wyrellis/IA-Embarquee/blob/master/pyfiles/vectorLearning/tenpointsvectorgenerator.py>

<sup>11</sup><https://github.com/wyrellis/IA-Embarquee/blob/master/pyfiles/vectorLearning/tenpointsvectorreader.py>

<sup>12</sup>[https://www.tensorflow.org/tutorials/load\\_data/numpy](https://www.tensorflow.org/tutorials/load_data/numpy)

<sup>13</sup>Librairie de structures mathématiques

<sup>14</sup><https://github.com/wyrellis/IA-Embarquee/blob/master/pyfiles/vectorLearning/tenpointsvectorIA.py>

<sup>15</sup>`labels = ['ASC', 'DESC']`

Une fois nos données formatées et mises dans des tableaux NumPy, nous pouvons directement passer à la gestion de notre réseau de neurones.

La configuration suivante est tirée de l'exemple donné sur les ressources en ligne sur TensorFlow<sup>16</sup>, cette configuration semble bien fonctionner avec les données tirées de structures de données simple NumPy. On modifie la configuration pour qu'elle puisse s'appliquer à notre exemple, on pose une première couche de 16 neurones (qui permettent d'effectuer la prise de décision), puis une seconde de 2 neurones, qui correspondra à la sortie. La compilation du modèle est également nécessaire à son fonctionnement.

Une fois configuré, le réseau est prêt à l'apprentissage. Le principe d'apprentissage repose sur le fait d'apprendre au réseau de neurones à associer un type de courbe à un nom de type de courbe (aussi appelé label). Pour cela, on génère un nombre assez conséquent de graphiques qui lui seront donnés. Le réseau de neurones va modifier chacun de ses neurones en fonction du graphique et du label associé, et ce, autant de fois qu'il y a de graphiques à apprendre.






















Nom	Modifié le	Type	Taille
 vector-0-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-1-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-2-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-3-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-4-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-5-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-6-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-7-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-8-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-9-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-10-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-11-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-12-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-13-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-14-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-15-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-16-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-17-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-18-ASC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-19-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko
 vector-20-DESC.csv	08/01/2020 10:57	Fichier CSV	1 Ko

Figure 5 : Visuel des éléments soit 1000 pour l'apprentissage

<sup>16</sup>[https://www.tensorflow.org/tutorials/load\\_data/numpy](https://www.tensorflow.org/tutorials/load_data/numpy)



Le réseau prendra en compte une liste NumPy de graphiques et une liste NumPy de labels (formatés comme décrit précédemment, et les deux listes doivent avoir une taille équivalente, puisque TensorFlow associe les deux éléments par leur index dans les listes).

On peut utiliser la fonction TensorFlow `.fit()` sur le modèle pour cela. Cette dernière prend aussi en paramètre le nombre d'itérations de l'apprentissage, cela à pour effet de 'rabâcher' les données et d'améliorer les résultats du réseau. TensorFlow appelle ces itération les 'Epochs'

```
Train on 2000 samples
Epoch 1/13
2000/2000 [=====] - 1s 628us/sample - loss: 0.5128 - sparse_categorical_accuracy: 0.6875
Epoch 2/13
2000/2000 [=====] - 0s 74us/sample - loss: 0.3309 - sparse_categorical_accuracy: 1.0000
Epoch 3/13
2000/2000 [=====] - 0s 66us/sample - loss: 0.1639 - sparse_categorical_accuracy: 1.0000
Epoch 4/13
2000/2000 [=====] - 0s 71us/sample - loss: 0.0596 - sparse_categorical_accuracy: 1.0000
Epoch 5/13
2000/2000 [=====] - 0s 72us/sample - loss: 0.0188 - sparse_categorical_accuracy: 1.0000
Epoch 6/13
2000/2000 [=====] - 0s 70us/sample - loss: 0.0051 - sparse_categorical_accuracy: 1.0000
Epoch 7/13
2000/2000 [=====] - 0s 74us/sample - loss: 0.0013 - sparse_categorical_accuracy: 1.0000
Epoch 8/13
2000/2000 [=====] - 0s 71us/sample - loss: 2.7600e-04 - sparse_categorical_accuracy: 1.0000
Epoch 9/13
2000/2000 [=====] - 0s 71us/sample - loss: 6.3600e-05 - sparse_categorical_accuracy: 1.0000
Epoch 10/13
2000/2000 [=====] - 0s 72us/sample - loss: 1.7754e-05 - sparse_categorical_accuracy: 1.0000
Epoch 11/13
2000/2000 [=====] - 0s 72us/sample - loss: 5.1236e-06 - sparse_categorical_accuracy: 1.0000
Epoch 12/13
2000/2000 [=====] - 0s 72us/sample - loss: 1.5814e-06 - sparse_categorical_accuracy: 1.0000
Epoch 13/13
2000/2000 [=====] - 0s 74us/sample - loss: 5.6529e-07 - sparse_categorical_accuracy: 1.0000
500/1 - 0s - loss: 3.1614e-07 - sparse_categorical_accuracy: 1.0000
Test with one vector of the set:
[[1.  0.75 0.75 0.625 0.5  0.25 0.25 0.  0.  0.  ]]
[[5.8892527e-07 9.9000096e-01]]
DESC
```

Figure 6 : Visuel de la phase d'apprentissage répartie sur plusieurs 'Epochs'

En dernier lieu, on procède à un test, on fournit au réseau de neurones d'autres graphiques et labels, mais c'est à son tour de nous dire à quel label ce graphique appartient. Un pourcentage de réussite sera donné à la fin. (image montrant le pourcentage de réussite). On a aussi décidé de faire quelques tests individuels, où l'on peut voir l'estimation que le réseau porte sur les labels ASC et DESC.

```

Vector n°1:
Graphique (sous forme de liste)
[[0.          0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
  0.55555556 0.77777778 1.          1.          ]]
Résultat du réseau [ASC,DESC]
[[9.8186111e-01 4.4962962e-06]]
Label choisi (par défaut, celui qui le meilleur résultat)
ASC
Vector n°2:
Graphique (sous forme de liste)
[[0.          0.11111111 0.22222222 0.33333333 0.44444444 0.66666667
  0.77777778 0.77777778 0.88888889 1.          ]]
Résultat du réseau [ASC,DESC]
[[9.8582226e-01 3.3522290e-06]]
Label choisi (par défaut, celui qui le meilleur résultat)
ASC
Vector n°3:
Graphique (sous forme de liste)
[[1.          0.88888889 0.77777778 0.77777778 0.55555556 0.44444444
  0.33333333 0.22222222 0.11111111 0.          ]]
Résultat du réseau [ASC,DESC]
[[2.1314747e-07 9.4558895e-01]]
Label choisi (par défaut, celui qui le meilleur résultat)
DESC

```

Figure 7 : Test du réseau de neurones

Le réseau de neurones est ici fonctionnel, nous avons dû trouver le nombre optimal de graphiques à apprendre et d'itérations d'apprentissage pour faire fonctionner au mieux notre réseau de neurones.

## 1.4 Travail sur la carte

### 1.4.1 Création d'un projet sous STM32CubeMX

Le logiciel est téléchargeable gratuitement sur internet, directement sur le site de ST<sup>17</sup>. Afin de créer un projet sous STM32CubeMX, nous avons suivis certains tutoriels<sup>18</sup> trouvés sur internet. On commence donc par sélectionner la carte via la section "board selector", en entrant NUCLEO-F767ZI. Puis on obtient la page "pins view" sur laquelle on peut effectuer les changements des pins, leur attribuer un rôle ou encore modifier leur nom afin de les reconnaître plus facilement lors des manipulations. Sur le microprocesseur non modifié, on obtient la configuration des pins suivante :

<sup>17</sup><https://www.st.com/en/development-tools/stm32cubemx.html>

<sup>18</sup>[https://www.waveshare.com/wiki/STM32CubeMX\\_Tutorial\\_Series](https://www.waveshare.com/wiki/STM32CubeMX_Tutorial_Series)

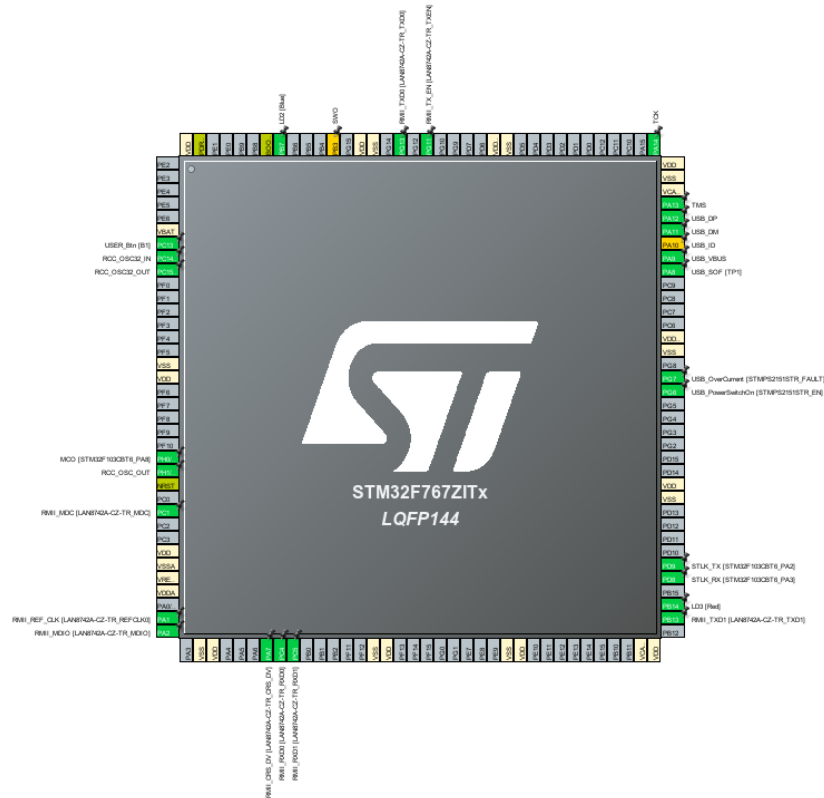


Figure 8 : Visuel du microprocesseur sans aucune modification

### 1.4.2 Configuration

Nous devons ensuite nous intéresser a la "clock configuration", qui prendra dans le champs HCLK(MHz) la valeur 216.

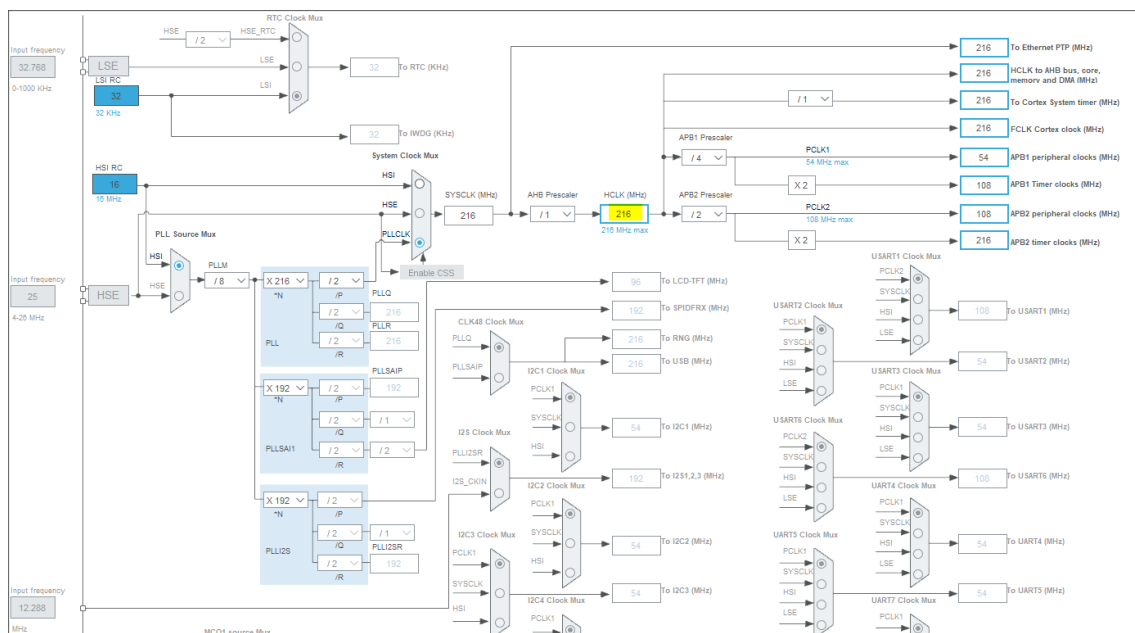


Figure 9 : Visuel de la clock

Nous pouvons ensuite choisir une configuration plus précise pour chaque pin.

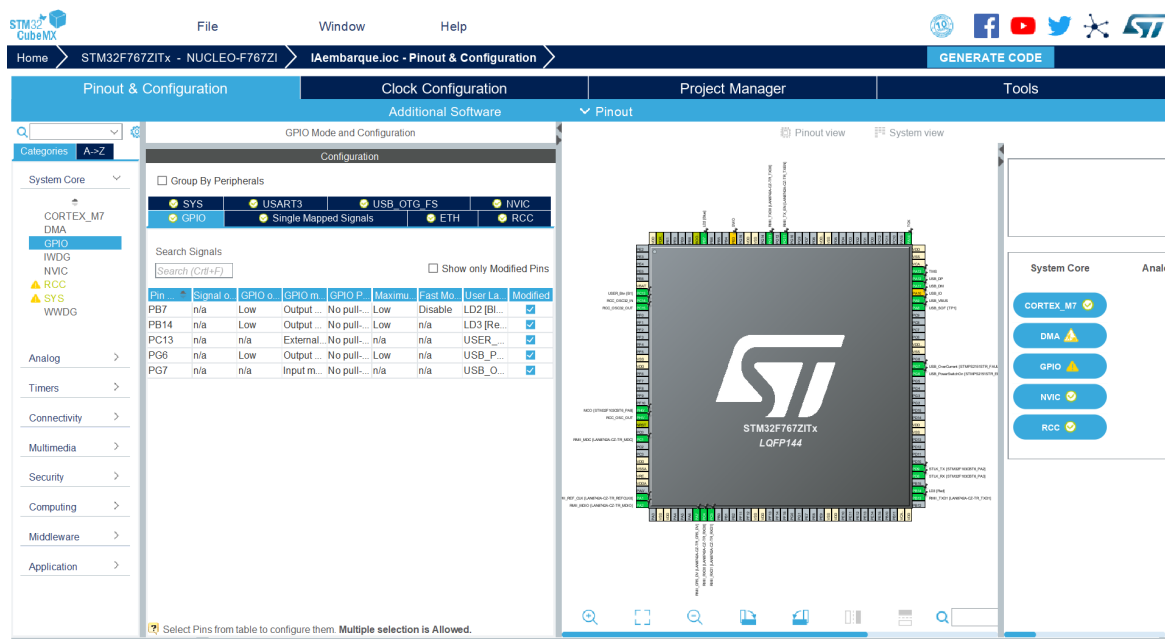


Figure 10 : Visuel du menu de modification du microprocesseur

Enfin, nous pouvons générer le code du projet dans le langage souhaité tout au long des modifications.

### 1.4.3 X-CUBE-AI

En ce qui concerne l'extension X-CUBE-AI<sup>19</sup>, nous l'avons téléchargé sur le site de ST<sup>20</sup>. C'est un jeu d'outils de développement de réseaux neuronaux qui s'intègre totalement dans le logiciel STM32CubeIDE, il y a seulement des cases à cocher pour signaler son utilisation dans le projet. Dans notre cas, cette extension peut nous permettre d'ajouter un réseau de neurones (keras, TFlite, Lasage, Caffe, ONNX, ...). Ici, nous utiliserons TensorFlow Lite, qui livre des fichiers `.tflite`<sup>21</sup>

<sup>19</sup><https://www.st.com/en/embedded-software/x-cube-ai.html>

<sup>20</sup>[https://www.st.com/content/st\\_com/en.html](https://www.st.com/content/st_com/en.html)

<sup>21</sup><https://github.com/wyrellis/IA-Embarquee/blob/master/pyfiles/voltGraphLearning/reseau.tflite>

## 2 Cycle 2

### 2.1 Acquisition de courbes

Les modèles de carte utilisées étaient un Arduino Duemilanove<sup>22</sup>, une STM32 MB997B et une Microship PICDEM 2 PLUS DEMO BOARD<sup>23</sup>. Les deux premières étaient alimentées par un générateur avec 5 volts et la dernière avec 9 volts. A l'aide de fils et d'un oscilloscope, nous avons récupéré la tension en fonction du temps et avons enregistré la courbe sous forme de fichier .CSV.

Pour obtenir assez de données, nous avons répété plus d'une centaine de fois le démarrage de la carte pour chaque modèle. Il aura fallu démarrer l'acquisition avec le même retard et le même échantillonnage pour chaque carte.

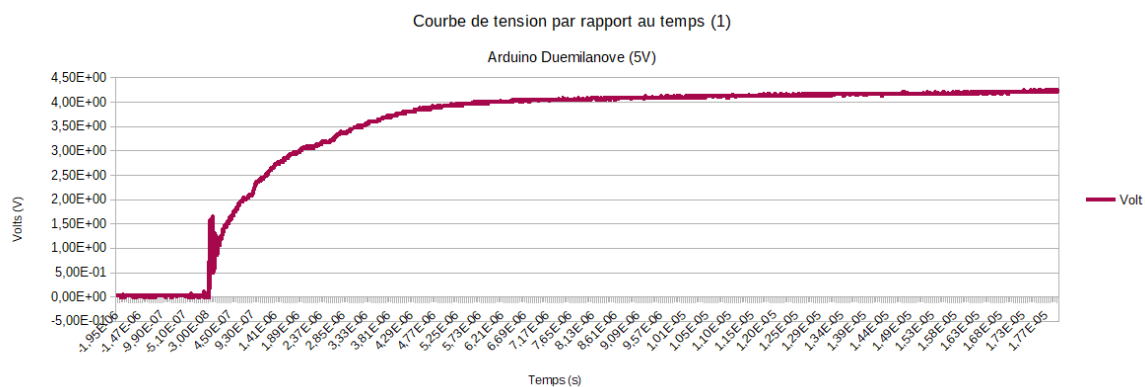


Figure 11 : Graphique acquis grâce à l'oscilloscope

### 2.2 Standardisation du format des données pour le Machine Learning

Pour faire un réseau de neurones qui distingue les types de carte grâce à leurs courbes de mise en tension, nous avons décidé d'extraire les données dans une durée prédéterminée et de commencer avec 100 points.

<sup>22</sup><https://www.arduino.cc/en/Main/arduinoBoardDuemilanove>

<sup>23</sup><https://www.microchip.com/Developmenttools/ProductDetails/DM163022-1>

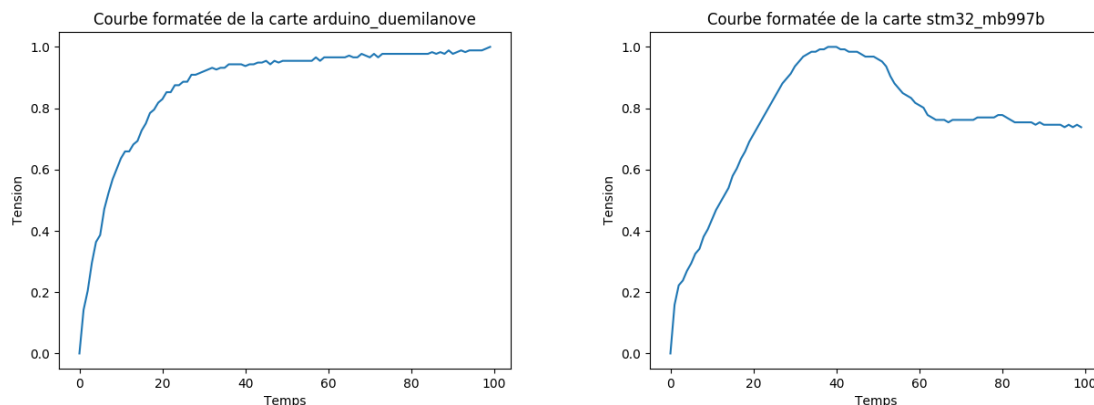


Figure 12 : Graphiques formatés appris par le réseau

On peut observer la différence entre les deux courbes de la figure 11, avec seulement la partie intéressante de la courbe (démarrage au seuil, signature visible).

Il faudra pondérer le nombre de cartes à apprendre, de sorte à ce qu'il n'y ait pas de déséquilibre d'apprentissage (que le réseau n'apprenne pas plus un même modèle de carte qu'un autre), ce qui réduira le nombre de courbes à apprendre au final, mais donnera un meilleur résultat suite à une utilisation sur le long terme.

### 2.3 Différenciation entre plusieurs cartes

Pour le moment, l'IA arrive à classer 3 cartes différentes grâce à leurs courbes de mise en tension. Le taux de confiance que l'IA attribué à sa déduction reste très faible (avec des taux avoisinant les 3% la plupart du temps).

```
Test n°1
[[0.03320981 0.01167783 0.00018156]]
true label :arduino_duemilanove
guessed label:arduino_duemilanove
Ici, 3% de confiance pour le arduino_duemilanove
Test n°2
[[3.2966327e-02 1.5122254e-01 9.8444198e-06]]
true label :microchip_picdem_2_plus_demo_board
guessed label:microchip_picdem_2_plus_demo_board
Test n°3
[[0.00960647 0.00072185 0.05845301]]
true label :stm32_mb997b
guessed label:stm32_mb997b
```

Figure 13 : Visuel d'un taux de déduction

Cependant l'IA arrive très nettement à écarter la possibilité de la mauvaise carte (taux de

confiance dix fois moins important que la bonne carte au minimum). Il faut aussi noter que la quantité de données d'entraînement utilisée est très petite (des lots de 150 fichiers) et que nous comparons 100 points, comparé aux nombres de vecteurs de 10 points récréés artificiellement (pour la première utilisation d'un réseau de neurones). Nous pouvons conclure sur le fait qu'au plus la carte apprend des données de courbes différentes, au mieux elle arrive à discerner les types de courbes.

## 2.4 Adaptation du réseau sur la carte

Ne sachant pas quel type de réseau de neurones nous devons implémenter dans notre carte, nous pensions que mettre un réseau TensorFlow pourrait fonctionner. Or, le module AI de l'IDE de STM32 nous autorise seulement Keras, TensorFlow Lite, et d'autres moins parlant. Keras étant un module utilisé dans la librairie TensorFlow 2.0, nous pensions que sauvegarder le réseau sous le Keras utilisé par TensorFlow pourrait fonctionner. Hélas, une erreur apparaît.

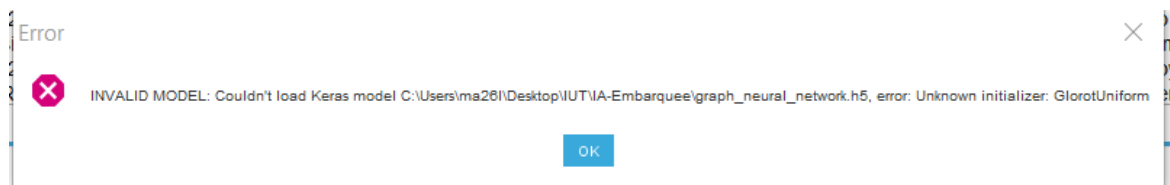


Figure 14 : Erreur apparue lors du transfert du réseau sur STM32CubeMX

Une solution serait de dissocier TensorFlow et Keras pour que cela puisse fonctionner, mais cela impliquerait que nous n'utilisions plus TensorFlow, ce qui nous aurait dérangé car c'est ce que nous pensions utiliser depuis le départ. Heureusement, TensorFlow 2.0 (version que nous utilisons) peut convertir les modèles en format exploitable par TensorFlow Lite. Le fichier binaire enregistré a pu être ouvert dans le module AI de l'IDE de STM32, avec les paramètres des couches de neurones renseignés depuis l'apprentissage.

## 3 Cycle 4

### 3.1 Implémenter le réseau de neurones

#### 3.1.1 Prise en main de la carte

Afin de comprendre le fonctionnement de la carte, nous avons lu sa documentation<sup>24</sup> et nous avons utilisé un multimètre<sup>25</sup>. Après avoir compris quelle pin était reliée aux User-Led et au User-Button, nous nous sommes penchés sur la création d'un programme qui ferait clignoter une LED définie.

Nous avons donc utilisé un tutoriel<sup>26</sup> trouvé sur internet.

---

<sup>24</sup>[https://www.st.com/content/ccc/resource/technical/document/user\\_manual/group0/26/49/90/2e/33/0d/4a/da/DM00244518/files/DM00244518.pdf/jcr:content/translations/en.DM00244518.pdf](https://www.st.com/content/ccc/resource/technical/document/user_manual/group0/26/49/90/2e/33/0d/4a/da/DM00244518/files/DM00244518.pdf/jcr:content/translations/en.DM00244518.pdf)

<sup>25</sup>Un multimètre (parfois appelé contrôleur universel) est un ensemble d'appareils de mesures électriques regroupés en un seul boîtier, généralement constitué d'un voltmètre, d'un ampèremètre et d'un ohmmètre.

<sup>26</sup><https://medium.com/@rlamarr/introduction-to-stm32cube-blinking-an-led-61469168f9e4>