

# Rapport technique Projet 2A

Idris Aggairi, Antoine Bernstein, Emmanuel Duguin,  
Pablo Dumenil, Margot Laleu, Théophile Mazerolle

April 29, 2022

## 1 Introduction - Antoine B.

Nous avons été déstabilisés lors des premières séances par le fonctionnement de ce groupe de projet basé sur l'innovation et n'avons pas tout de suite saisi l'importance de la phase de réflexion, alors que c'est elle qui nous mène à innover. Nous avions une vision plutôt précise de ce que nous voulions faire et avons mis du temps avant de partir vers d'autres idées, que l'on espère plus innovantes. Ce premier rapport cependant abordera l'idée avortée et l'idée finale car cette première nous a permis de nous renseigner d'un point de vue technique mais aussi d'apprécier le marché et la concurrence.

## 2 Première idée : Le Carré - Margot L.

A l'issue de la première séance où nous nous sommes individuellement présentés ainsi que nos idées de projet, ce qui nous a regroupés tous les 6, c'est l'envie de faire de la musique et d'innover. Nous étions tout d'abord partis sur l'idée d'Antoine, Idris et Théophile, à savoir un launchpad en stand-alone. L'idée partait d'un constat. Lorsque l'on souhaite faire de la musique à l'aide d'un launchpad, il faut forcément le connecter à un ordinateur, ce qui rend son transport difficile. De plus, la première chose à faire est d'apprendre à se servir d'un logiciel complexe tel qu'Ableton.

Nous voulions donc concevoir un launchpad dont l'utilisation est simplifiée, que l'on utilise uniquement à l'aide d'une application smartphone, et qui communiquent ensemble sans fil, donc sûrement en bluetooth. Nous voulions pour simplifier notre launchpad une interface comportant quasiment aucun bouton en dehors des pads répartis selon une matrice carrée, d'où son nom : "Le Carré".

Nous avions commencé à réfléchir aux technologies à utiliser (bluetooth, prise jack, etc), au fonctionnement de l'application et à la priorisation et une première répartition des tâches à affiner au fur et à mesure de l'avancée du projet. Nous avions récapitulé tout cela dans le tableau ci-dessous.

Features	Priorité	Nom
Codage : jouer une note	1 A.I	
Récupérer le boîtier : pièces détachées	1 I	
Design minimal de l'appli	1 E.T.M	
L'appuie déclenche le son	1 A.I	
led allumée quand ON - Pads rétroéclairage	1 A	
Association sample-pad	1 T.M	
Créer des boucles	2	
coder la config du pad en une variable 4x4	1	
gestion des enceintes	2 T+M	
bluetooth	2	
gestion des bibliothèques	2 P	
sauvegarder un signal dans la mémoire de la carte et le jouer	1	
sauvegarder une mélodie dans le téléphone	3	
mémoriser des sets déjà créés	2/3	
liaison app - hardware	3	
switch ON/OFF	1 A.I	
sortie jack	1 E	
mode clavier	2	
Compléter les bibliothèques audio	3 P	

Nos priorités étaient donc de trouver un support physique préexistant sur lequel on pourrait au moins faire nos tests dans un premier temps, puisque créer notre propre support comme on le souhaitait prendrait sûrement trop de temps. Et il fallait en parallèle réfléchir aux fonctions et visuel minimal de l'application pour commencer à coder le fonctionnement, à savoir faire glisser les samples d'une bibliothèque vers une case de la matrice correspondant aux pads de notre instrument.

Finalement, après étude du marché, nous nous sommes rendu compte que nos idées étaient relativement proches de l'existant. Nous avons donc utilisé des méthodes de réflexion individuelle puis en groupe faisant partie des méthodes de Brainstorming dont l'objectif est de stimuler la créativité spontanée de chaque membre du projet, puis nous avons commencé une réflexion sur les moyens d'interagir avec un instrument de musique à travers nos 5 sens. Ce qui nous a conduits à notre projet actuel.

### 3 Nouveaux objectifs - Antoine B.

Après l'idée du "carré" avortée, nous avons recommencé une réflexion qui est partiellement décrite dans cette partie.

La ligne conductrice du projet peut se résumer en une phrase : "Permettre une production musicale intuitive". Sur les moyens de production numérique de musique actuelle il y a en général une quantité vertigineuse de réglages et de personnalisations possible permettant de moduler le son. Mais tous les avoirs en tête et réussir à trouver les meilleurs dans une situation donnée s'avèrent très vite compliqués, d'autant plus pour un débutant. D'où l'organisation de notre projet autour de l'intuitivité. Avant de rentrer plus dans le détail de notre réflexion, nous tenons à indiquer que nous ne cherchons pas à faire un instrument plus "facile" mais bel et bien plus "intuitif", dans le sens où les intentions du musicien seront les plus fidèlement retranscrites tout en gardant un niveau de réglage et de personnalisation aussi important qu'avant.

Notre recherche a commencé par la reconsideration des sens qui pourraient être utiles à la production musicale. Si le goût et l'odorat ne nous ont pas menés loin, il en est autrement pour le toucher. Lors de la composition à l'aide d'instruments numériques le toucher est utilisé, mais seulement dans un sens: de l'utilisateur à l'instrument. C'est en cela que la musique numérique s'oppose à celle des instruments traditionnels : il n'y a pas de retour de l'instrument vers l'utilisateur. Jouer de la guitare c'est aussi sentir les vibrations de cordes, différencier les cordes graves et aiguës par leur diamètre. C'est pour cela que nous voulons implémenter une forme de retour haptique sur notre instrument.

Autre point qui différencie le numérique du "réel" : la continuité du milieu dans lequel nous évoluons. L'utilisation de touches qui chacune représente une note est un frein à l'intuitivité selon nous. Notre instrument sera donc une surface, avec des marquages pour se repérer mais chaque point de cette surface sera sensible et aura un effet différent de son voisin. Cette surface aura aussi une autre particularité : elle sera souple dans le but de permettre une modulation selon la force de l'appui. Ce dernier point donne un aspect organique à notre instrument, toujours dans la volonté de le rendre intuitif.

Enfin, dernier point clé de notre projet : la reconnaissance de geste sur la surface et l'interprétation de ceux-ci pour composer. Comme expliqué plus haut, il existe une combinaison de réglages inimaginable lorsque l'on compose et trouver les réglages adaptés à notre composition peut s'avérer être extrêmement laborieux et long. Notre instrument sera donc capable de retranscrire les émotions que le musicien veut faire passer dans sa composition simplement par le biais des réglages adaptés, et cela simplement en effectuant un geste caractérisant les émotions en question sur la surface de l'instrument.

Ainsi, après étude des possibilités, nous avons choisi de réaliser un instrument avec un pad unique. Le musicien peut alors effectuer un geste dessus qui se traduira, via une IA, par un réglage spécifique d'un VST (synthétiseur virtuel). Une fois ce VST réglé, le musicien pourra jouer avec ce synthétiseur via une autre interface indépendante de notre instrument.

Un second objectif serait de faire de notre instrument, un instrument à part entière, c'est-à-dire un instrument sur lequel on peut régler les paramètres de notre VST, mais aussi jouer avec ce VST réglé. Le musicien aurait alors un retour haptique via le Touchpad sur lequel il joue.

Ce développement nous permet d'en venir aux différentes parties autour desquelles s'articulera notre projet :

- Partie matériaux et capteurs.
- Partie traitement des données des capteurs.
- Partie I.A. pour l'interprétation de certaines de ces données.

Celles-ci sont développées dans les parties qui suivent.

## 4 Le choix des matériaux - Idris A.

Après une phase d'étude de marché comparative, et un questionnement sur le ressenti physique que nous voulions pour notre contrôleur. Il nous est apparu que la texture permettant le plus d'expressions sensitives était celle du latex ou du silicone. En effet, ce ressenti complexe mêlant douceur et rugosité, selon comment on agit dessus, nous a semblé répondre tout à fait aux besoins que nous avions.

Nous avons donc commencé nos essais avec du silicone de type plomberie, que l'on retrouve en quincaillerie. Ces matériaux étaient vendus en tubes d'assez petite contenance, et à un prix plutôt élevé compte tenu de la quantité. De plus leur texture s'est révélée être trop pâteuse et visqueuse pour pouvoir la couler correctement : on obtenait des surfaces inégales et accidentées qui ne répondaient pas au cahier des charges.

Après avoir essuyé ce premier échec, nous avons donc décidé d'étudier plus sérieusement les caractéristiques des matériaux de type latex. Ils sont classés sur une échelle : l'échelle Shore, qui se décline en plusieurs versions selon la dureté recherchée. L'illustration ci-dessus permet de comprendre ce classement :

Pour l'usage que nous souhaitons en faire, la texture recherchée possède une dureté située entre les indices 20 et 40 de l'échelle Shore A (entre l'élastique et la gomme). Nous nous sommes donc procurés sur Internet, un pot de silicone indexé Shore A 30, qui semblait particulièrement indiqué pour la réalisation de moules.

Pour se rendre compte des échelles de dureté, et des processus de moulage, nous conseillons le visionnage de la vidéo suivante, en français :

<https://youtu.be/AJOKgdnBmM0>

## 5 Le choix des capteurs - Emmanuel D.

### 5.1 Une première approche

Nous avions trois capteurs qui sont à étudier afin de retenir le plus adapté aux besoins du projet. Ces capteurs sont :

- un capteur piézoélectrique : Produit un courant électrique fonction de la dérivée de la force appliquée
- un capteur piezo-capacitif : Production d'une tension en fonction de la force exercée
- capteur de contrainte résistif : Variation de la résistance en fonction de l'effort appliqué

### 5.2 Approche expérimentale

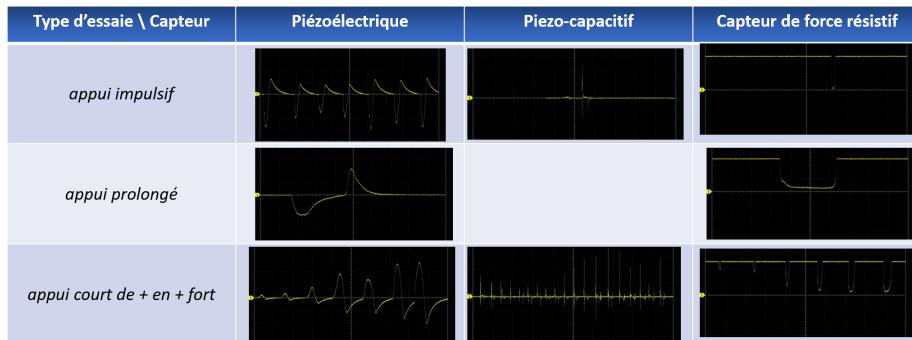
Afin de comprendre et connaître le comportement de chacun d'eux, nous avions tout d'abord utilisé un pendule de test (masse en plomb reliée à un axe par un fil). Ce dispositif nous permettait d'itérer nos relevés, à l'aide d'un oscilloscope, sur les différents capteurs de manière précise pour une meilleure comparaison.

Etant donné que notre produit présente une composante artistique, il était plus judicieux d'opérer les relevés en fonction de la sensation perçue par l'utilisateur, c'est-à-dire procéder à trois types de relevés que sont

- un appui impulsif
- un appui prolongé
- des appuis présentant une intensité croissante.

On a visualisé les signaux générés par ces capteurs. Pour le capteur piézoélectrique ainsi que le piézo-capacitif, nous avions directement branché les dipôles aux bornes de l'oscilloscope. Une variante pour le piézoélectrique était de le mettre dans un circuit intégrateur. Pour le capteur résistif, nous l'avons placé dans un circuit pont diviseur de tension afin d'observer la variation de résistance via la tension aux bornes du capteur.

Un tableau reprenant ces différentes configurations est présenté ci-dessous :

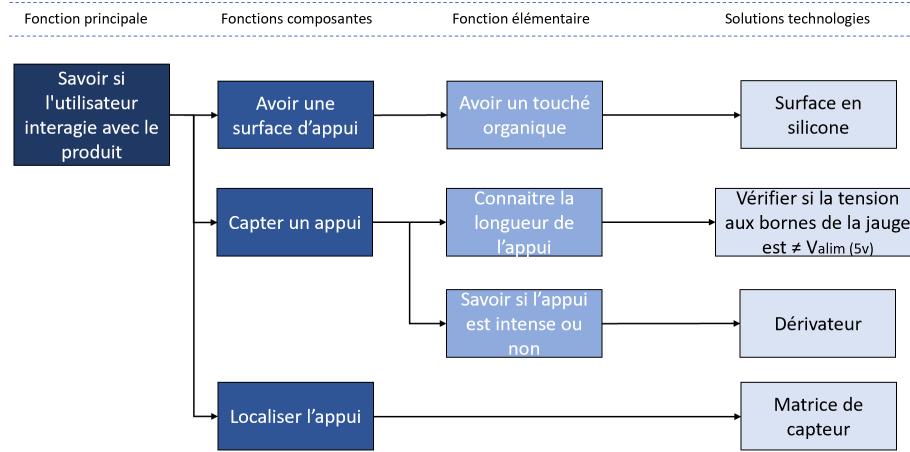


A la fin de cette expérimentation, un tableau récapitulant l'ensemble des avantages et défauts de chaque capteur a été synthétisé dans le tableau suivant :

	Résistif	Piézoélectrique	Piezo capacatif
Avantage	-Plage de tension fixe (entre 0 et $V_{\text{alim}}$ ) -tension à ses bornes linéaires en fonction de la force exercée	-bonne insertion à une surface plane -Sensible (aux variations) -peu cher	-Plage de tension pas très élevé (quelques centaines de mV)
Inconvenant	-Peu sensible pour un toucher léger (-> pont de Wheatstone) -Moyenne insertion par répétition (mise en juxtaposition) -plus onéreux	-Plage de tension non bornée (pouvant dépasser la 20ème de V) -Sensible (aux variations) (retour haptique compromis) -Impulsionnel, tension tend à s'annuler lors d'un appui long	-Mauvaise insertion (branchement)

A ce stade, le capteur de contrainte résistif semble être un bon choix vis-à-vis de nos besoins, le capteur piezo-capacitif étant écarté de par sa très grande difficulté d'insertion dans le projet (qui est d'ailleurs cause de l'absence d'un relevé). Néanmoins d'autres pistes de capteurs pourraient être à prendre en compte comme le velostat.

Ainsi nous pouvons réaliser une partie du diagramme FAST qui concerne la fonction principale "Savoir si l'utilisateur interagit avec le produit".



L'idée étant, par la suite, de disposer les capteurs sur un support afin d'y constituer une matrice de signaux analysables par une IA.

### 5.3 Etude d'un nouveau matériau

Après avoir recherché quel capteur serait le mieux adapté à notre projet, nous nous sommes penchés sur un nouveau matériau qu'est le Velostat (ou linqstat) qui possède le même comportement qu'une résistance variable (tout comme le capteur de contrainte résistif), à la différence qu'il se présente sous la forme d'un film conducteur, beaucoup plus intéressant que le placement d'une multitude de capteurs les uns à côté des autres. Ce qui, par ailleurs, nous permettra d'avoir une meilleure résolution sur la captation d'un appui. Il semble donc être un excellent choix au vu des besoins de notre projet, à savoir obtenir une bonne précision dans la localisation de l'appui pour une meilleure interprétation des gestes par intelligence artificielle plus tard.

Ce matériau a la particularité d'avoir une valeur de résistance qui diminue à mesure que la force que l'on va lui appliquer à sa surface augmente.



Le velostat présente une double résistivité : surfacique (entre deux points sur l'une des deux surfaces) et volumique (entre les deux surfaces). La résistance la plus intéressante pour notre projet étant la résistance volumique. Nous verrons par la suite comment nous allons utiliser ce matériau afin de répondre à nos besoins.

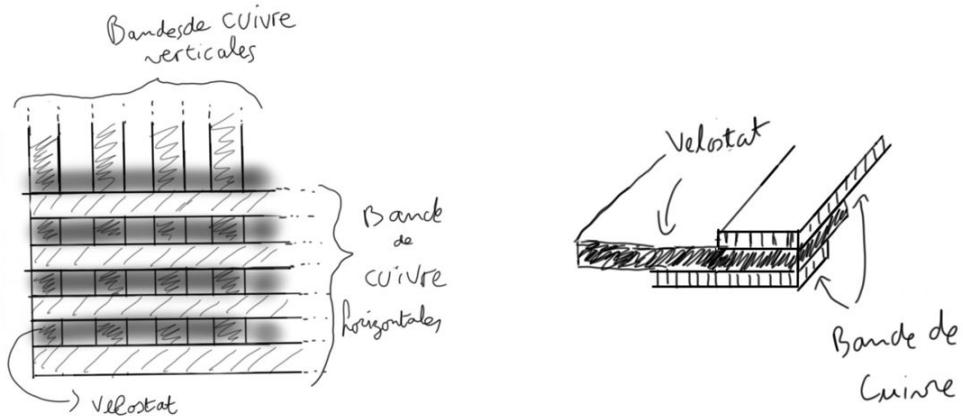
### 5.4 Mise en oeuvre

Il s'agit à présent de pouvoir utiliser les propriétés du velostat afin de répondre à une de nos contraintes, à savoir "localiser l'appui". Pour ce faire, nous allons réaliser un maillage en 3 couches :

- une première couche composée de bandes de cuivre disposées verticalement et à intervalles réguliers.

- une couche composée du velostat.
- une dernière couche composée également de bandes de cuivre disposées horizontalement et à intervalles réguliers.

Par la suite, nous appellerons ce maillage le PAD (en référence à sa structure planaire) qui constitue le point de départ de l'interaction Homme - Machine (HM) de notre produit.



Le principe de ce PAD est de pouvoir, par la suite, l'insérer dans un circuit électronique nous permettant d'avoir une image de la position du toucher, et ce avec une résolution  $n^*n$ ,  $n$  étant le nombre de bandes de cuivre utilisé. Chaque point d'intersection entre les bandes de cuivre horizontales et verticales constituant un point de cette matrice de points.

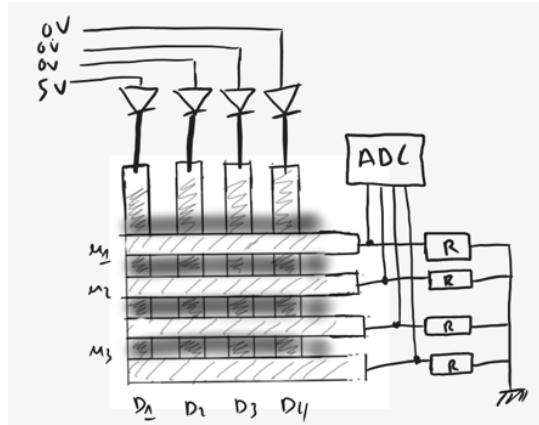
## 6 Circuit électronique et principe de fonctionnement

Dans cette partie, nous allons voir le circuit électronique réalisé afin de pouvoir y insérer notre PAD.

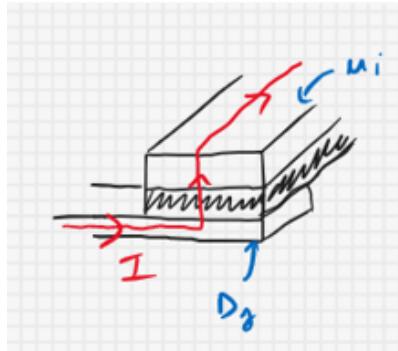
### 6.1 Principe de fonctionnement

Nous allons utiliser la propriété de résistance variable du velostat, afin de mesurer la tension aux bornes de chaque point du quadrillage du PAD, par un pont diviseur de tension.

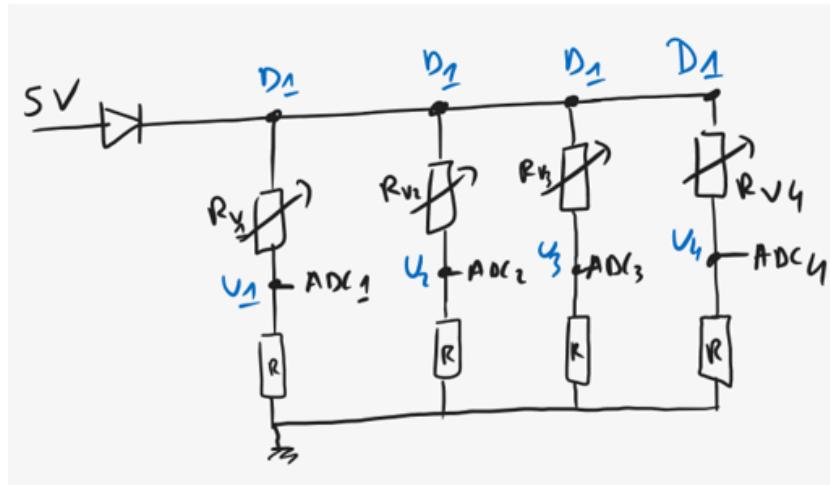
Pour ce faire, nous utiliserons des résistances fixes dont la valeur sera à déterminer par la suite. Pour mieux comprendre le fonctionnement de l'ensemble, nous allons nous intéresser au schéma suivant.



Chaque bande de cuivre verticale (qu'on appellera simplement colonne) va être successivement alimentée par une tension de référence (5V), tandis que les autres seront à 0V. On repérera ainsi chaque point par ses coordonnées ( $U_i, D_j$ ), respectivement pour Up et Down, qui correspondent à la ligne  $i$  et la colonne  $j$ .  $i$  et  $j$  appartenant à l'intervalle  $[0, n]$ . Par exemple, considérons que la colonne n°1 est alimentée (cf.  $D_1$ ). Alors, le courant circulant dans celle-ci passera par chaque point ( $U_i, D_1$ ) du velostat pour ensuite sortir par les  $N$  résistances fixes  $R$ . On place ainsi un ADC (Analogue to Digital Converter ou Convertisseur Analogique Numérique) sur chaque sortie de ligne pour obtenir un pont diviseur de tension.



Le schéma équivalent pour l'alimentation de la première colonne est représenté sur la figure suivante.



Les  $R_{Vi}$  représentent les résistances variables générées par le velostat aux points ( $U_i, D_1$ ). L'utilisation de diodes est essentielle afin d'éviter un courant de fuite dans les colonnes adjacentes.

## 6.2 Choix des résistances fixes

Il faut à présent déterminer la valeur des résistances fixes, qui sera choisie dans l'optique d'avoir la plage de tension maximale aux bornes de l'ADC. Pour ce faire, il faut au préalable connaître la plage de variation de la résistance variable introduite par le velostat à un point quelconque de la matrice. Expérimentalement, on mesure cette plage qui va de 170 Ohm pour un appui très important, à 1,7k Ohm pour un état de repos (sans appui). On utilise par la suite un tableur qui, pour un grand nombre de valeurs de résistances, nous donne les tensions maximales et minimales obtenues avec celles-ci et ainsi la plage de tension optimale, en considérant l'intervalle de résistivité d'un point du velostat. La figure ci-dessous représente ce tableau.

Rv	R	Vmax	Vmin	plage de tension	Max plage de tension
170	50	1,14	0,14	0,994	
1700	100	1,85	0,28	1,574	
	150	2,34	0,41	1,938	
	200	2,70	0,53	2,176	
	250	2,98	0,64	2,335	
	300	3,19	0,75	2,441	
	350	3,37	0,85	2,512	
	400	3,51	0,95	2,556	
	450	3,63	1,05	2,583	
	500	3,73	1,14	2,595	
	550	3,82	1,22	2,597	
	600	3,90	1,30	2,592	
	650	3,96	1,38	2,580	
	700	4,02	1,46	2,565	
	750	4,08	1,53	2,545	
	800	4,12	1,60	2,524	
	850	4,17	1,67	2,500	
	900	4,21	1,73	2,475	
	950	4,24	1,79	2,449	
	1000	4,27	1,85	2,422	
	1050	4,30	1,91	2,394	
	1100	4,33	1,96	2,366	
	1150	4,36	2,02	2,339	
	1200	4,38	2,07	2,311	
	1250	4,40	2,12	2,283	
	1300	4,42	2,17	2,255	
	1350	4,44	2,21	2,228	
	1400	4,46	2,26	2,201	
	1450	4,48	2,30	2,174	
	1500	4,49	2,34	2,147	
	1550	4,51	2,38	2,121	
	1600	4,52	2,42	2,096	
	1650	4,53	2,46	2,070	

On constate que la valeur de résistance nous permettant d'avoir la plage de tension maximale aux bornes de l'ADC est  $R = 550$  Ohm avec  $V_{\text{min}} = 1,22\text{V}$  et  $V_{\text{max}} = 3,82\text{V}$  soit une plage de tension de quasiment 2,6V.

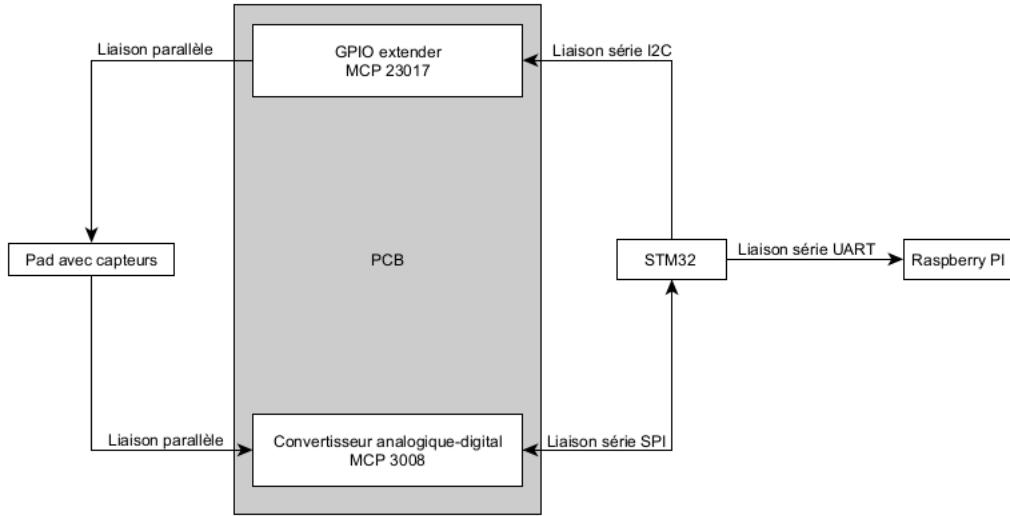
#### Remarque :

Les convertisseurs analogique-numérique nécessitent deux tensions que sont la tension d'alimentation et une tension dite de référence qui influe sur la résolution (ou quantum) de l'ADC. Dans notre prototype, nous avons utilisé une tension de référence égale à la tension d'alimentation, soit 5V. Après réflexion, cette valeur de tension n'est pas judicieuse car elle ne nous permet pas d'avoir une résolution maximale et ne nous permet donc pas de visualiser au mieux les variations de pressions liées à l'appui de l'usager. Or, ce calcul a été réalisé en considérant une tension  $V_{\text{ref}}$  de 5V, ce qui n'est pas tout à fait le cas de par l'utilisation de diodes qui introduisent une chute de tension d'environ 0,7V et qui vont venir diminuer encore  $V_{\text{max}}$ . Cela ne change pas la valeur de résistance R à prendre en compte que nous avons calculée, néanmoins, il serait préférable de tenir compte de la chute de tension des diodes et ainsi de prendre comme tension de référence non pas 5V mais la valeur de la tension maximale théorique (moyennant une marge de 0,1) qui sera appliquée à l'entrée du convertisseur.

## 7 Récupération des données et communication - Antoine B.

### 7.1 Schéma général

Le schéma suivant décrit la manière dont les différents composants communiquent entre eux :



Les communications entre le Pad et le PCB sont analogiques et celles entre le PCB, la STM et la Raspberry sont numériques.

### 7.2 Echanges entre le pad et le PCB - Emmanuel D

Nous avons évoqué la nécessité de plusieurs ADC afin de mesurer la tension aux bornes de chaque résistance fixe R de sortie, par un pont diviseur de tension. Au total, il en faut autant que le nombre de lignes (/colonne) dont dispose notre PAD. Pour notre prototype, nous avons choisi une matrice 16\*16, soit une résolution de 256 pixels d'information. Il faut donc 16 ADC. Notre choix s'est porté sur 2 mêmes puces offrant chacune 8 ADC 10bits, que sont les MCP3008. Nous verrons leur fonctionnement plus loin.

De plus, il faut pouvoir imposer successivement une tension haute (5V) sur chaque colonne du PAD. La tension par défaut étant l'état bas (0V). Pour cela, notre choix s'était porté au départ sur un registre à décalage, une technologie qui fonctionne mais trop ancienne, quand on sait que la fonction à réaliser peut se faire simplement par un GPIO Extender tel que le MCP23017. C'est donc ce dernier que nous avons décidé de prendre comme composant de commande.

### 7.3 Echanges entre le PCB, la STM32 et la Raspberry

Pour avoir le fonctionnement décrit dans la partie précédente, il faut envoyer des messages aux MCP3008 et MCP23017. On utilise pour cela la carte de développement STM32F746-G. La communication se fait par liaison série (I2C pour le MCP23017 et SPI pour le MCP3008). Sur la STM, la commande de l'envoi des messages sur les différentes liaisons se fait par interruption grâce à un timer. A chaque fois que le timer déclenche une interruption, l'ensemble des 256 points de la matrice de capteurs vont être lus (via deux boucles "for" avec 16 itérations). Voici le code exécuté à chaque interruption :

```

void TIM6_DAC_IRQHandler(void)
{
    extern char spi_buf[256];

    extern const uint8_t MCP23017_A[16];
    extern const uint8_t MCP23017_B[16];

    for (int k=0;k<16;k++){

        hmcp.gpio[MCP23017_PORTB] = MCP23017_B[k]; //
        hmcp.gpio[MCP23017_PORTA] = MCP23017_A[k]; //Put 1 line at 5V and all
        mcp23017_write_gpio(&hmcp, MCP23017_PORTA); //the others at 0V
        mcp23017_write_gpio(&hmcp, MCP23017_PORTB); //

        for (int p=0;p<16;p++){
            spi_buf[16*k+p] = read_ch_MCP3008(p); //read the value for each point of the line of sensors
        }
    }

    send_matrix_uart_1(); //send the data to the raspberry
}

```

Les fonctions permettant l'utilisation du MCP23017 via une liaison I2C proviennent d'un github public ([github.com/ruda/mcp23017](https://github.com/ruda/mcp23017)). Pour le MCP3008, nous avons réalisé les fonctions de lecture à partir des fonctions "HAL". Elles sont regroupées dans le fichier functions.c sur notre Git. A la fin de chaque interruption, la fonction "send matrix uart 1()" est exécutée et envoie les données à la Raspberry. (Pour plus de détails sur le code, les fichiers sont tous sur le Git il y a une vidéo explicative : video)

Il suffit maintenant de récupérer les données sur la Raspberry, de les stocker dans une matrice et de les envoyer à l'IA pour traitement.

## 8 Prototypage - Antoine B.

### 8.1 Les éléments à évaluer

Après avoir déterminé les types de capteurs potentiels dont nous aurions l'usage, ainsi que la matière de la surface du contrôleur, la question de l'intégration conjointe de ces éléments s'est posée.

Nous en sommes venus à la conclusion que différents paramètres étaient à évaluer : tout d'abord, le ressenti au toucher de la surface, puis les signaux émis par les capteurs coulés dans la surface. Quelle configuration, avec quels capteurs, nous permettrait de récupérer le plus d'information, le plus fidèlement possible ?

### 8.2 Des paramètres clés

Pour répondre à cette question, nous avons décidé de faire varier indépendamment les uns des autres, plusieurs facteurs clés :

- L'épaisseur de silicium
- Le type de capteur utilisé (évoqué en 5.)
- L'espace inter-capteur dans la matrice
- La disposition de ces capteurs (matrice carrée, circulaire, etc.)

### 8.3 Tests à réaliser

Nous avons pris la décision de déterminer dans un premier temps l'épaisseur de silicone et le type de capteur à utiliser. Pour cela nous avons réalisé trois moulages de dimensions 8x8 cm, contenant chacun un capteur piézo et un capteur résistif centrés, pour des épaisseurs de silicone de 0.5cm, 1cm et 2cm. Cela nous a permis d'évaluer plusieurs ressentis digitaux, et de comparer les réponses des différents capteurs.

### 8.4 Réalisation du moule

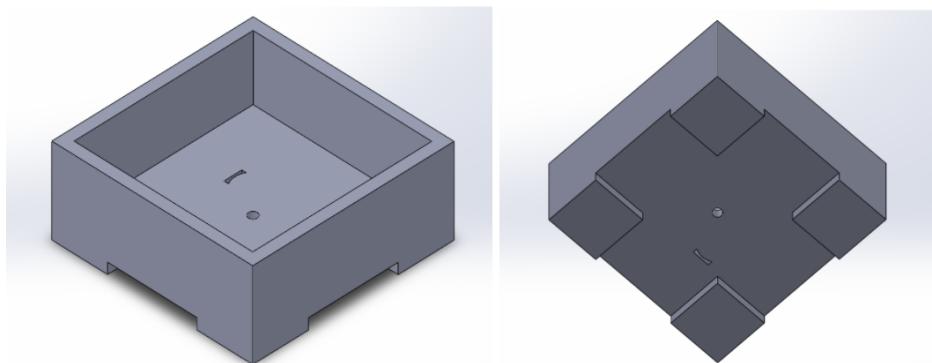
#### 8.4.1 Moule v1

Le moule de test est destiné à être répliqué de nombreuses fois pour les besoins des phases de tests (notamment au regard du nombre de paramètres à tester). Il doit donc répondre au cahier des charges suivant :

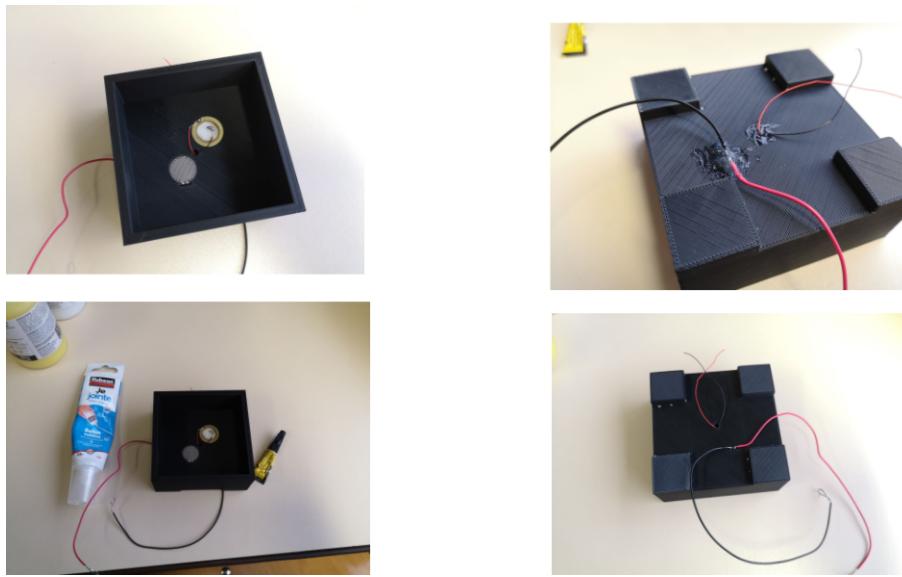
Le moule doit être :

- Simple (pour être répliqué facilement)
- Modulable (pouvoir contenir tous types de capteurs)
- Permettre une étanchéité (pour couler le silicone après placement des capteurs)
- Réduit (pour consommer moins de silicone par test)

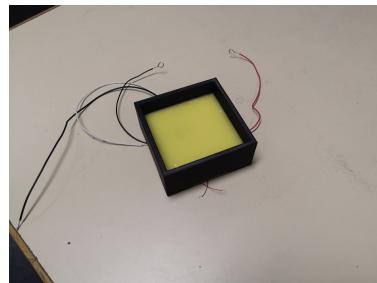
Son design final a donc été déterminé sur le logiciel SolidWorks dans le respect des critères précédents :



On remarque l'existence de deux types de trous : ils ont été réalisés d'après les mesures des composants à insérer et sont donc parfaitement adaptés au positionnement des capteurs et de leurs nappes respectives. Après impression 3D, les capteurs ont été fixés avec de la colle, et les trous comblés avec le silicone de joints de nos premiers essais.

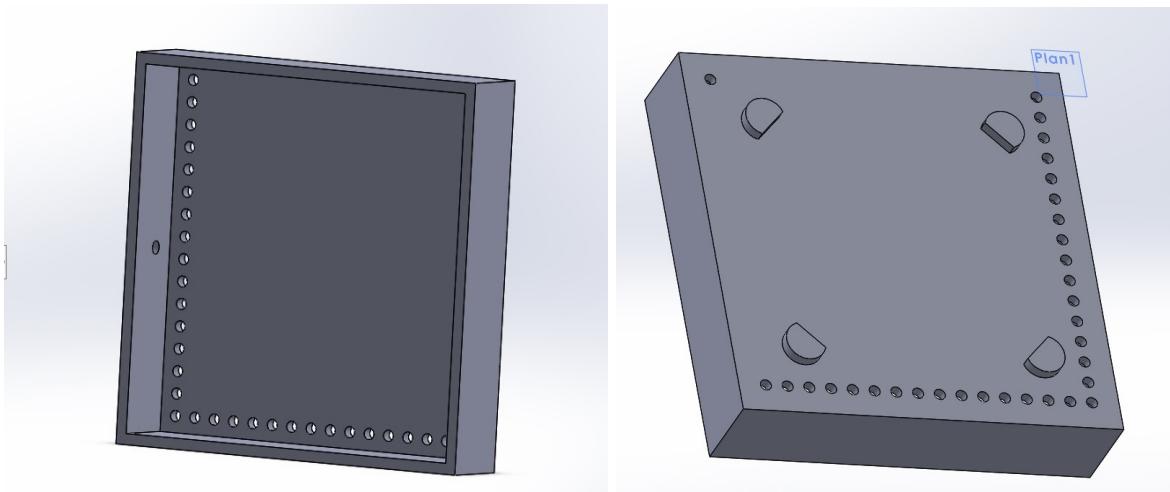


Enfin, on ajoute le silicone :

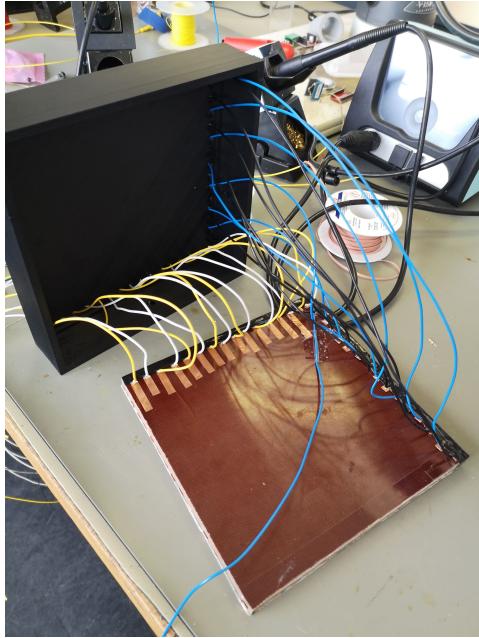


#### 8.4.2 Moule v2

La première version du moule avait pour vocation de permettre le test de différents capteurs. Ce second moule est lui destiné à être un prototype utilisable. Il est ainsi plus grand (une quinzaine de centimètres de côté) et comporte les trous nécessaires au câblage de la matrice de capteurs fabriquée à partir du vélostat.



Dans ce nouveau moule nous avons inséré la plaque, sur laquelle la matrice de capteurs se trouve, et fait passer tous les fils par les trous prévus à cet effet :



Ce moule est muni d'un trou supplémentaire sur un de ses côtés pour faire passer les fils des vibrateurs dans le silicone, et ses pieds ont une forme permettant de coller des capteurs piézoélectriques dessus et ainsi avoir une meilleure précision d'un point de vue dynamique.

## 9 Intelligence Artificielle

### 9.1 Pourquoi une I.A. ? - Théophile M.

Le choix de l'intelligence artificielle s'est fait naturellement de par les caractéristiques du projet : nous voulons traduire l'intention du musicien à travers la manière dont il interagit avec la surface de notre instrument. Pour cela nous relevons un champ de pression qui est stocké dans une matrice. Nous avons en fait une image dont la valeur des pixels est une valeur de pression. Il faut alors associer cette image à des pré-réglages sur un synthétiseur. Le procédé est identique à de la reconnaissance d'image et va permettre au musicien de s'exprimer librement.

Pour mettre en place cette I.A, plusieurs étapes sont nécessaires :

- L'apprentissage
- Une première IA de type multicouches
- L'IA finale de type CNN

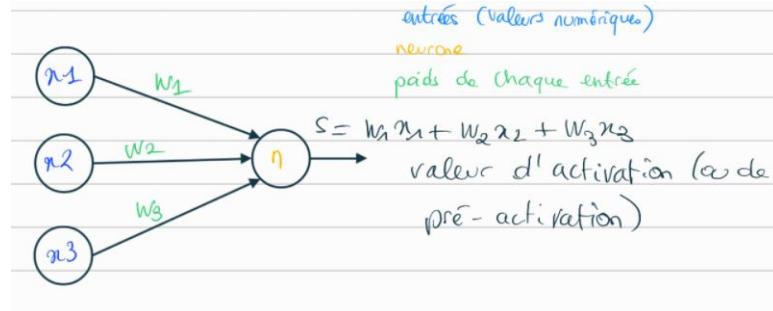
### 9.2 L'apprentissage - Théophile M.

Tout d'abord, il a fallu définir le fonctionnement d'une IA :

Une IA fonctionne grâce à des neurones, plus particulièrement un réseau de neurones. Ce réseau est composé de layer, qui sont des étages comportant plusieurs neurones. Un neurone prend plusieurs données en entrée et y associe un poids. Il somme les produits poids\*donnée pour obtenir une valeur. Si cette valeur dépasse le seuil prévu, le neurone active la sortie à laquelle il est associé.

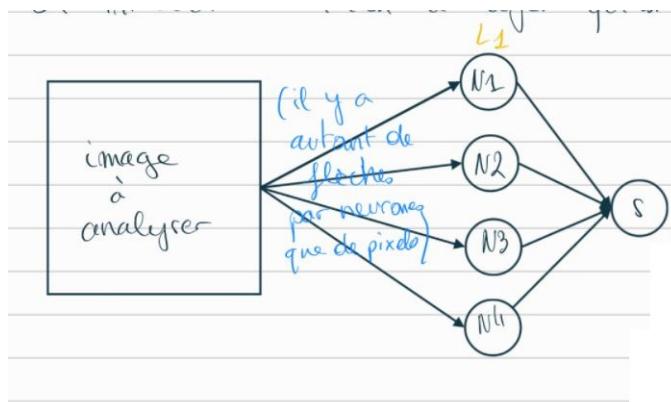
Le Perceptron ou le neurone

Le neurone seul est appelé perceptron.



Dans un réseau de neurones, le premier layer va donc récupérer les données initiales et les analyser. Les prochains layers vont ensuite analyser les données qui sortent du premier layer, jusqu'à atteindre le dernier layer qui donnera le résultat.

Réseau de neurones à 1 seul layer



Une fois le réseau créé, nous devons l'entraîner avec une banque de données d'images. Il va alors se corriger avec la méthode de descente de gradient. Pour cela, l'erreur entre la valeur d'activation et la valeur de sortie est calculée. Le gradient des erreurs est calculé et les poids sont alors corrigés pour que la valeur de sortie tende vers la bonne valeur.

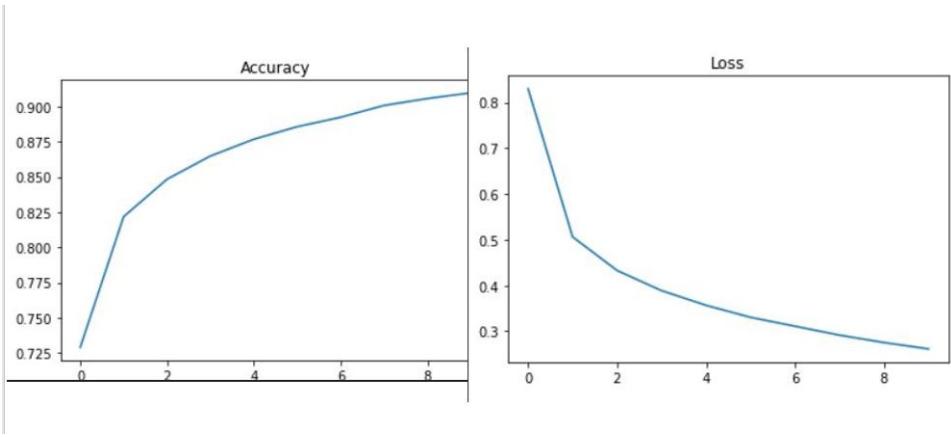
### 9.3 Une première IA - Théophile M.

Nous avons alors créé une première intelligence artificielle simple pour comprendre comment implémenter un tel système. Cette première intelligence artificielle permet de reconnaître 10 types de vêtements sur des images de 28x28 pixels. On utilise une base de données disponible via Keras (bibliothèque open source écrite en python). Nous avons créé 3 layers :

- Un premier layer comprenant 256 neurones
- Un second layer avec 128 neurones
- Un dernier layer constitué de 10 neurones (correspondant aux 10 sorties possibles)

Chacun des 256 neurones de la première couche est relié aux  $28 \times 28 = 784$  pixels de l'image. Chacun des 128 neurones de la seconde couche est relié aux 256 sorties de la première couche.

Nous avons entraîné le réseau et relevé ses courbes de progression :



On relève bien une amélioration des résultats. Sur ce simple réseau de neurones, nous atteignons des résultats corrects très rapidement.

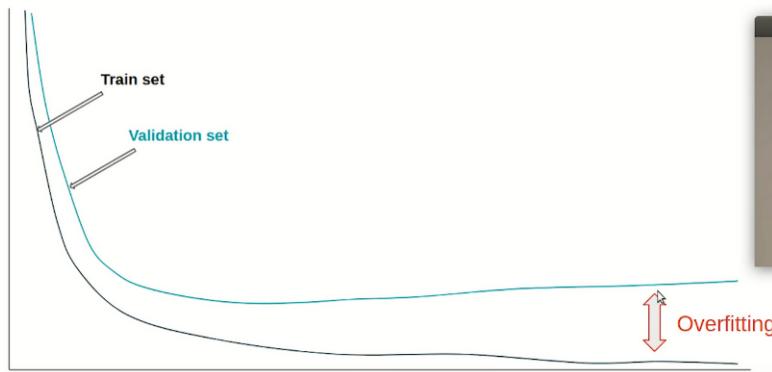
#### 9.4 Attention au sur-apprentissage (overfitting) - Margot L.

En effet, le problème qui se pose lors de l'apprentissage de notre modèle, c'est qu'à partir d'un moment, la banque d'images étant toujours la même, le modèle va s'attacher à des détails des images qui ne sont pas caractéristiques de ce qu'il doit apprendre à reconnaître. Par exemple, s'il doit reconnaître des chats et que ces derniers sont parfois dans un salon, alors le modèle peut identifier le canapé comme étant caractéristique du chat et il sera alors moins apte à reconnaître un chat dans un autre environnement non présent dans la base de données d'apprentissage.

Cela aura pour cause, un taux d'erreur bien supérieur à celui que l'on observe sur la base de données d'apprentissage, lorsqu'on lui présentera de nouvelles images.

Le but va donc être d'éviter ce sur-apprentissage. Nous allons donc utiliser pour cela les trois jeux suivants:

- Jeu d'apprentissage (train set)
- Jeu de validation (validation set)
- Jeu de test

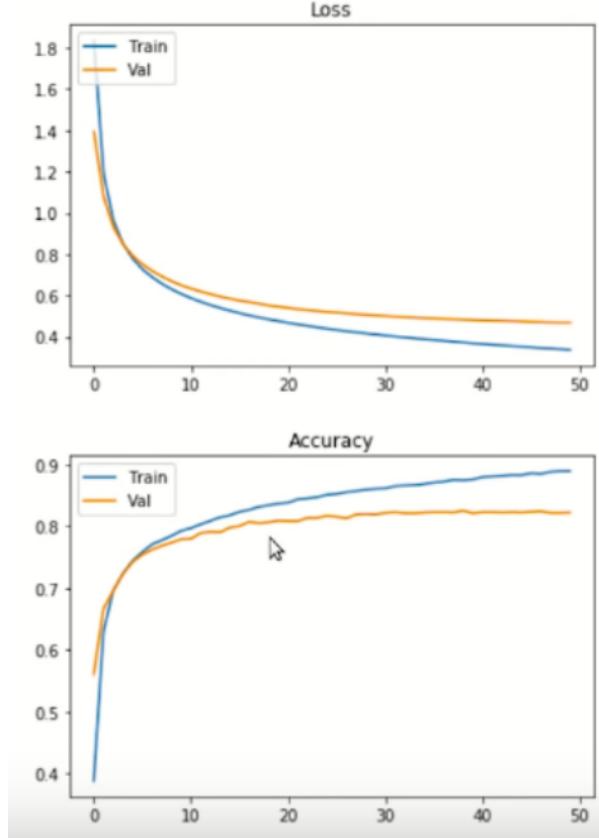


Notre réseau de neurones s'entraîne toujours sur le jeu d'apprentissage. Sauf qu'à présent, entre deux passages du jeu d'apprentissage dans le réseau, le jeu de validation va passer mais lui n'impliquera aucune modification du réseau et permettra d'observer l'évolution de l'efficacité réelle de l'apprentissage au cours de l'apprentissage.

Une fois l'apprentissage terminé, le jeu de test permet de tester l'état d'apprentissage de notre réseau.

Mais pour que cela fonctionne, il faut que les données dans ces trois jeux soient suffisamment différentes, sinon on observerait, par le même mécanisme, un sur-apprentissage. Par exemple, si on cherche à identifier des chats, il ne faut pas que des photos du même chat et/ou de la même pièce se retrouvent dans les différents jeux de données.

En testant cette méthode dans notre code on obtient les courbes ci-dessous.



On voit alors que la validation atteint un plateau, on n'est donc pas encore en sur-apprentissage, on peut donc décider d'arrêter l'entraînement maintenant.

## 9.5 Codage d'un réseau de neurones multicouches - Margot L.

Pour coder notre première IA, nous avons utilisé Google Colab qui permet de coder en Python directement en ligne. Pour coder une IA il s'agit donc d'installer TensorFlow qui est une plate-forme open source regroupant plusieurs outils dont une bibliothèque Python permettant de coder une IA.

Nous installerons également la bibliothèque Scikit-learn, nommée sklearn, permettant également de coder un réseau de neurones, responsable de l'apprentissage automatique.

Il nous faut ensuite importer différentes bibliothèques telles que : matplotlib.pyplot pour l'affichage de courbes, numpy pour la gestion d'opérations mathématiques et des matrices. Ensuite pandas pour l'analyse de données. StandardScaler lui permet de normaliser les valeurs des entrées pour optimiser la descente de gradient (on centre et réduit chaque valeur).

```
[ ] import tensorflow as tf
print(tf.__version__)

2.8.0

▶ import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sys

from sklearn.preprocessing import StandardScaler
```

Pour cet exemple de réseau de neurones multicouches nous allons utiliser une dataset d'images de vêtements appartenant à 10 classes différentes. Cette dataset est disponible dans la bibliothèque keras de TensorFlow.

La dataset importée, comporte les images ainsi que d'autres informations associées. Dans un premier temps, on crée 2 matrices, l'une dans laquelle on stocke les images et l'autre les targets associés. Les targets sont un nombre (allant de 0 à 9) servant d'identifiant à la catégorie à laquelle appartient le vêtement sur l'image associée.

Ensuite, on n'a pas besoin de travailler sur l'entièreté de la dataset, on va donc réduire les matrices précédentes à 30 000 valeurs.

L'étape suivante consiste à aplatisir les matrices et en normaliser les données.

Aplatir consiste à ce qu'une image de (28, 28) devienne un vecteur de  $28 \times 28 = 784$ .

```
# flatten
images = images.reshape(-1, 784)
images = images.astype(float) # on s'assure que les variables sont des flotants

# on crée l'objet qui normalise, c'est à dire qui centre par rapport à la moyenne et réduit.
# Puis on l'utilise pour normaliser notre base de données
scaler = StandardScaler()
images = scaler.fit_transform(images)
```

Afin de limiter l'overfitting, on divise notre dataset en deux jeux, un jeu d'apprentissage, et un jeu de test.

Une fois cela fait, il nous reste à coder notre modèle lui-même. C'est-à-dire initialiser un modèle dont on choisit le type, ici de type séquentiel. Le type séquentiel sert à ce que les opérations du modèle soient exécutées les unes après les autres, donc chaque opération a pour entrer le résultat de l'opération précédente.

Et ensuite, on ajoute les opérations dans notre modèle, dans l'ordre d'exécution. Nos opérations consistent en les couches successives, paramétrées comme on le souhaite.

```
model = tf.keras.models.Sequential()

model.add(keras.layers.Dense(256, activation="relu"))
# L'opération Dense sert à créer une couche de neurones,
# le premier argument est le nombre de neurones de la couche.
model.add(keras.layers.Dense(128, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

# Le softmax fait en sorte que la somme de tous les neurones de sortie
# soient égale à 1. On a alors une distribution de probabilité.
```

Pour finaliser le modèle, il faut encore le compiler et définir sa fonction d'erreur :

```

model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="sgd",
    metrics=["accuracy"]
)

```

Le loss contient la fonction d'erreur choisie en fonction du modèle utilisé.

Le sparse-categorical-crossentropy sert, dans un premier temps, à sélectionner, parmi les 10 pourcentages en sortie dus à notre softmax, celui correspondant à la target attendue. L'objectif, va donc être de maximiser ces valeurs, donc de les faire tendre vers 1 puisque ce sont des probabilités. Pour cela, sparse-categorical-crossentropy va prendre le log de ces valeurs afin de converger plus vite. En prenant ensuite l'opposé de ces valeurs, l'objectif de ce modèle sera de minimiser les valeurs ainsi obtenues.

Le mot sparse de notre fonction d'erreur signifie que l'on peut prendre un chiffre comme targets, au lieu de la matrice des 10 sorties avec un 1 à l'indice correspondant à la classe de l'image.

L'optimizer choisi ici permet de minimiser l'erreur en utilisant la descente de gradient. Et "accuracy" signifie que les taux de prédictions justes seront affichés en pourcentage.

Le modèle est donc prêt, on peut en visualiser un résumé :

```

model.summary()
# Le nombre de paramètres qu'il affiche correspond au nombre de
# paramètres modifiables par tensorflow pour réduire l'erreur en sortie.

Model: "sequential"

Layer (type)                 Output Shape              Param #
=====
dense (Dense)                (None, 256)             200960
dense_1 (Dense)               (None, 128)              32896
dense_2 (Dense)               (None, 10)               1290
=====
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0

```

Et il est donc possible de l'entraîner avec la commande ci-dessous.

```
history = model.fit(images, targets, epochs=10)
```

On reconnaît en premier et deuxième arguments la matrice contenant les images et la matrice contenant les targets associés. Le troisième argument, indique le nombre d'epochs, c'est-à-dire le nombre de fois que notre réseau de neurones va s'entraîner sur notre dataset fournie.

Une dernière étape, consiste à sauvegarder notre modèle déjà entraîné, et ensuite pouvoir le recharger grâce à h5py.

## 9.6 Choix du type de réseau de neurones - Margot L.

Finalement, nous nous sommes rendu compte qu'il existe différents types ou différentes structures de réseaux de neurones. Celui détaillé précédemment est de type multicouche. Si on reste sur ce modèle, ce

qu'il s'agit de dimensionner c'est le nombre de couches cachées du réseau ainsi que le nombre de neurones par couche, ce qui a une influence sur la qualité de l'apprentissage.

Nos recherches nous ont indiqué que le type de réseau le plus adapté à notre situation, c'est-à-dire pour des images de grande taille et plus complexes serait le CNN (réseau neuronal convolutif) bien adapté pour les images (2D) mais aussi les volumes (3D) ce qui pourrait notamment permettre d'intégrer un paramètre supplémentaire.

Si nous avions voulu travailler en temporel par exemple, donc avec une entrée qui varie dans le temps, on parle alors de réseau neuronal à retard temporel (TDNN).

Si le modèle multicouche fonctionne aussi pour le traitement d'images, il devient inadapté pour des images de grande taille puisque le nombre de connexions entre neurones augmente exponentiellement puisque chaque neurone est alors relié à tous les neurones de la couche précédente et de la couche suivante.

Les réseaux de neurones convolutifs limitent ce nombre de connexions.

L'une des caractéristiques de ce type de réseau est qu'il est invariant par translation, c'est-à-dire que le motif détecté par ce noyau est indépendant de la localisation spatiale dans l'image.

Nous décidons donc d'utiliser un réseau de neurones convolutif (CNN), puisque nous n'avons pas besoin de la localisation du geste.

## 9.7 Le CNN en théorie - Margot L.

Le CNN analyse l'image de manière plus précise, notamment en repérant des motifs plus complexes.

En multicouche : on aplatis l'image et on la traite en entier. Cependant quand les images sont trop grandes, le réseau de neurones n'est plus capable de les traiter, c'est pourquoi on utilise un réseau à Convolution.

### 9.7.1 Filtres

Un CNN utilise des filtres afin de réduire le nombre de données simultanées à analyser.

On utilise alors des filtres (Kernel) qui sont des matrices ayant une taille donnée bien inférieure à celle de l'image initiale. Cette matrice va venir se multiplier avec une petite partie de l'image, soit quelques pixels, et fournir en sortie un seul pixel.

Par exemple, une matrice de  $5 \times 5$  analysera 25 pixels de l'image initiale et les remplacera par un seul pixel. Et cette matrice va ainsi se déplacer sur la totalité de l'image. On crée ainsi une image contenant moins de pixels, donc moins de données.

Lors de l'entraînement de l'IA, ce seront les poids des matrices qui seront modifiés, de la même manière que les poids d'un réseau de neurones multicouches. En modifiant ainsi les valeurs des matrices, elles seront capables de reconnaître des motifs spécifiques. Par exemple, une matrice qui se spécialise à reconnaître les contours de l'objet.

On peut ainsi utiliser plusieurs filtres, successivement, qui analyseront différents motifs, détails de l'image.

De cette manière, chaque filtre va créer une nouvelle image.

Ainsi, par exemple, si on a une image initiale de  $x \times y$ , alors on aura une image en sortie de la somme des  $N$  filtres de  $a \times b \times N$ .

La convolution est donc le fait de faire passer un filtre sur la totalité de l'image et a pour conséquence d'augmenter la profondeur de l'image en sortie.

Une fois une série de filtres passés sur l'image initiale et une image plus petite mais plus profonde obtenue, on dit qu'on a effectué 1 convolution. Il est possible voire nécessaire de recommencer l'opération sur les images (ou les couches de l'image) précédemment obtenues.

Ainsi, les convolutions successives vont avoir pour conséquence de réduire la taille de l'image, ce qui limitera le nombre de convolutions que l'on pourra faire, car il arrivera un moment où l'on ne pourra plus réduire l'image. Mais aussi de permettre à l'IA de s'intéresser à des détails de plus en plus précis.

### 9.7.2 Pooling

Il existe différents types de pooling.

Prenons l'exemple du max pooling, qui sert encore une fois à réduire l'image précédemment convoluée, mais cette fois en conservant uniquement le pixel ayant la valeur maximale de ceux analysés, toujours sur le principe d'une matrice qui se déplace sur l'image. Ainsi, on va diminuer le nombre de calculs ensuite nécessaires et ainsi économiser du temps de calcul et de la mémoire.

Il est important de noter que le pooling efface des données, celles qui sont liées à la localisation du motif dans l'image.

Mais cela à l'avantage de limiter l'overfitting. En effet, comme le réseau de neurones perd l'information de la localisation des détails dans l'image, il ne pourra pas apprendre une position précise de l'objet dans l'image, et donc ne pourra pas éliminer la présence d'un chat dans l'image alors que le chat est seulement excentré dans l'image par rapport aux autres images.

### 9.7.3 Synthèse

Ainsi, pour créer un réseau de neurones à Convolution, il s'agira d'alterner les couches de convolution et de pooling.

En sortie de ses différentes couches, on aura alors une image 3D, correspondant aux différents filtres utilisés et donc aux différents détails repérés.

Pour obtenir la sortie de notre réseau de neurones, il faut alors aplatisir cette matrice 3D de taille fortement réduite, et utiliser dessus un réseau multicouche classique.

## 9.8 Codage d'un réseau de neurones à convolution - Margot L.

Notre code a été importé sur la Raspberry, mais il a, à l'origine, été codé également sur Google Colab, et donc toujours en Python.

Beaucoup de choses dans le principe vont être similaires au codage du réseau multicouche. A commencer par Tensorflow et Sklearn à installer et plusieurs bibliothèques à importer :

```
7  import tensorflow as tf
8  from tensorflow.keras import layers
9  from sklearn.model_selection import train_test_split
10 from sklearn.preprocessing import StandardScaler
11 import pandas as pd
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import os
15 import pathlib
16 import sys
17 import numpy as np
18 import random
```

Une fois cela fait, l'étape suivante consiste à récupérer la dataset permettant d'entrainer notre IA, comportant donc des matrices 16\*16 et les targets associées.

Or, dans notre cas, nous n'avons pas pu constituer une dataset. C'est pourquoi, afin de vérifier le fonctionnement de notre IA, nous créons une dataset aléatoire grâce aux fonctions matGen et dataGen.

Cette partie du code sera donc à remplacer par l'import de la dataset obtenue depuis le drive de Google, d'où l'import de drive en début de code.

```
4   from google.colab import drive  
5   drive.mount('/content/drive')
```

L'étape suivante, afin de limiter l'overfitting, est de cinder notre dataset en un jeu d'apprentissage et un jeu de validation, ayant une taille respective de 80 pourcents et de 20 pourcents, indiqué par le troisième argument de la fonction train-test-split.

```
images, images_valid, targets, targets_valid = train_test_split(data, targets, test_size=0.2)
```

Ensuite, afin de se déplacer facilement dans la dataset, il est nécessaire de "l'officialiser", c'est comme dire à l'ordinateur, voilà ce qui constitue notre dataset. On le fait grâce à la fonction tf.data.Dataset.from\_tensor\_slices.

Ensuite, on établit le nombre d'epochs souhaité pour notre entraînement, ainsi que la taille de nos batchs à 32. Un batch est un sous-ensemble de notre dataset puisque l'ordinateur n'est pas capable de les traiter toutes simultanément. Quand on a parcouru l'ensemble de la dataset par batch, on a fait une epoch.

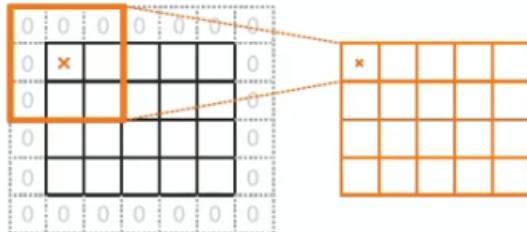
On peut à présent créer notre modèle. Comme précédemment, on crée un modèle de type séquentiel auquel on ajoute les étapes à réaliser dans l'ordre dans lequel on souhaite qu'elles soient exécutées.

```
79 model = tf.keras.Sequential([  
80     layers.experimental.preprocessing.Rescaling(1./255),  
81     layers.Conv2D(32, 4, padding='same', activation='relu'),  
82     layers.Conv2D(64, 3, padding='same', activation='relu'),  
83     layers.Conv2D(128, 3, padding='same', activation='relu'),  
84     layers.Flatten(),  
85     layers.Dense(128, activation='relu'),  
86     layers.Dense(num_param, activation='Softmax')  
87 ])
```

Notre première étape consiste à normaliser les éléments de la dataset afin de limiter les effets d'échelle, comme nous l'avions fait pour le réseau multicouche, sauf que nous l'avions antérieurement au modèle.

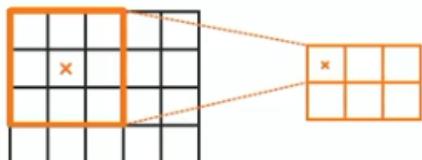
Ensuite, on ajoute les couches du réseau de neurones. Notre réseau est constitué de 3 couches de convolution. Par exemple, notre première couche de convolution est constituée de 32 filtres, chacun de taille 4\*4.

Sans information contraire en argument, la matrice constituant le filtre se déplacera sur l'image d'une colonne par une colonne lorsqu'elle se déplace horizontalement, puis d'une ligne par une ligne verticalement, une fois son déplacement sur l'horizontale actuelle effectué. L'argument padding sert à dire si l'on souhaite faire une sorte de zéro padding, c'est-à-dire ajouter un contour virtuel de zéro, comme on le voit ci-dessous afin de conserver la taille de l'image, ou si l'on préfère ne pas le faire et avoir une diminution de l'image au passage de chaque filtre et ainsi être limité en nombre de filtres en fonction de la taille initiale de l'image.



`padding = 'same'`

The use of a padding allows to keep the size of the image.



`padding = 'valid'`

Means no padding

Ensuite, il faut aplatisir notre matrice 3D obtenue en sortie de ces filtres de convolution. En effet, la différence est qu'en multicoche, on aplatis les images avant de les envoyer dans le réseau, ce n'est pas le cas dans un réseau convolutionnel puisque les calculs de filtres consistent en une multiplication terme à terme de matrices.

Or, après les couches de convolution, on termine par un réseau multicoche appliqué à notre sortie des couches de convolution. Ce réseau multicoche n'est ici constitué que d'une couche d'entrée et d'une couche de sortie comportant 8 sorties (données par num-param) pour notre premier prototype.

Ensuite, comme dans le réseau multicoche, on compile le modèle en définissant les fonctions de loss, optimizer et metrics que l'on souhaite. Ici l'optimizer 'adam' a aussi pour but de minimiser l'erreur sur la descente de gradient en utilisant une méthode stochastique.

```
89 model.compile(optimizer='adam',
90                  loss=tf.losses.CategoricalCrossentropy(from_logits=True),
91                  metrics=['accuracy'],)
```

model.fit sert alors à lancer l'entraînement de réseau en spécifiant la dataset, la présence d'un jeu de validation, le nombre d'epochs souhaité.

La dernière étape consiste alors à sauvegarder le modèle entraîné, à l'aide de `model.save(chemin-du-dossier- où-sauvegarder)`.

Remarque : Les fonctions `model.summary` et `tf.keras.callbacks.TensorBoard` servent respectivement à observer la structure du réseau de neurones et les statistiques de son fonctionnement, notamment sur le flux de données du réseau durant l'apprentissage.

## 9.9 IA et Raspberry

### 9.9.1 Préparation de la Raspberry Pi 4 8Go - Margot L.

Une Raspberry est un mini-ordinateur, la première chose à faire est donc de la connecter à un écran, une souris et un clavier et de lui insérer une carte SD. Avant cela, sur la carte SD, il faut télécharger et installer "Raspberry Pi Imager" qui sert ensuite à simplifier l'installation d'un OS sur la Raspberry qui en est à l'origine dépourvue.

Il faut faire attention à bien installer la bonne version de l'OS, entre Raspbian 32 bits ou 64 bits. Nous, il nous faut la version 64 bits.

L'étape suivante consiste à regarder les différentes versions des programmes préinstallés dont nous allons avoir besoin : TensorFlow, Python. Cela se fait en ligne de commande via les commandes montrées ci-dessous. Ce qu'il faut vérifier c'est qu'on a bien les mêmes versions que celles soulignées en rouge.

```

pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)" 1
NAME="Debian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
VERSION_CODENAME=bullseye
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.10.63-v8+ #1488 SMP PREEMPT Thu Nov 18 16:16:16 GMT 2021 aarch64 GNU/Linux 2
pi@raspberrypi:~ $ python3 --version
Python 3.9.2 3
pi@raspberrypi:~ $ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/aarch64-linux-gnu/10/lto-wrapper
Target: aarch64-linux-gnu
gcc version 10.2.1 20210110 (Debian 10.2.1-6)
pi@raspberrypi:~ $ pip3 list | grep numpy
numpy 1.19.5 4
pi@raspberrypi:~ $ 5

```

Il est ensuite nécessaire de télécharger Tensorflow et de l'installer directement en ligne de commande. Pour cela nous avons utilisé le code ci-dessous.

```

# get a fresh start
$ sudo apt-get update
$ sudo apt-get upgrade
# install pip3
$ sudo apt-get install git python3-pip

# or download wheel
$ git clone https://github.com/Qengineering/Tensorflow-io.git
$ cd Tensorflow-io
$ sudo -H pip3 install tensorflow_io_gcs_filesystem-0.23.1-cp39-cp39-
linux_aarch64.whl
$ cd ~

# install gdown to download from Google drive
$ sudo -H pip3 install gdown
# download the wheel
$ gdown https://drive.google.com/uc?id=1YpxNubmEL_4EgTrVMu-kYyzAbtyLis29
# install TensorFlow 2.8.0
$ sudo -H pip3 install tensorflow-2.8.0-cp39-cp39-linux_aarch64.whl

```

### 9.9.2 From tensor Slices - Théophile M.

Les données que le réseau CNN sera amené à traiter sont récupérées sous forme de tableau numpy de taille 16\*16. Nous avons donc utilisé tensorflow.data.Dataset.fromtensorslices() pour moduler la dataset afin qu'elle soit utilisable par tensorflow. Manquant de données, nous avons créé cette dataset arbitrairement à

partir de tableau numpy. Dans le cadre de la poursuite du projet, il faudrait bien évidemment générer cette dataset en utilisant notre prototype pour récupérer les données d'entrée, et les associer manuellement à des réglages du synthétiseur comme souhaités.

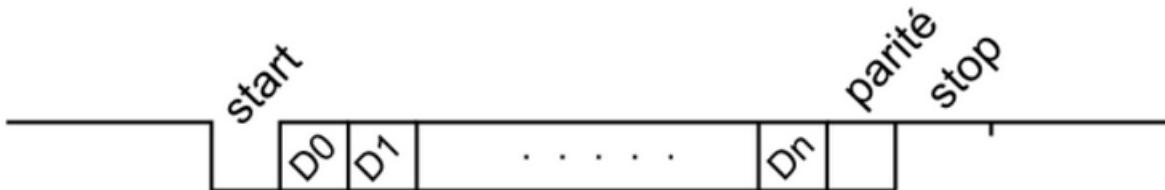
### 9.9.3 Les différentes liaisons utilisées avec la Raspberry - Théophile M.

La Raspberry est connectée par USB-UART avec la carte STM32 et envoie en UDP la sortie du réseau de neurones, qui correspond aux réglages des paramètres choisis du synthétiseur. La carte STM32, elle, récupère les données des capteurs dans le pad et les envoie par liaison UART-USB sur la Raspberry. Les données sont récupérées sur la Raspberry sous la forme d'un tableau numpy directement exploitable par notre code python utilisant l'IA entraînée.

### 9.9.4 Liaison UART - Margot L.

Une liaison UART (émetteur-récepteur asynchrone universel) fait partie des liaisons de type série qui permettent de relier deux systèmes numériques qui communiquent entre eux en échangeant des données bit à bit qui se succèdent, les uns après les autres, sur un canal de communication qui peut être filaire, radio voire optique.

La liaison UART est filaire et la trame envoyée est de la forme suivante :



- le niveau logique de repos (en l'absence de transmission) est le niveau logique HAUT, soit 1, défini par le niveau 5V ;
- la transmission est initiée par un bit de START à 0 qui sert à synchroniser l'envoi de la trame avec le récepteur ;
- suivent les bits de données dont la taille est comprise entre 5 et 9 bits qui sont envoyés du LSB (bit de poids faible) au MSB (bit de poids fort) ;
- suit un bit de parité pour le contrôle d'erreur ;
- suivent un ou deux bits de STOP (niveau logique HAUT, soit à 1, programmable par l'utilisateur) pour marquer la fin de la trame.

L'une des premières choses à faire est alors de synchroniser les deux côtés de la liaison. Il faut initialiser la liaison du côté émetteur et du côté récepteur en réglant notamment la vitesse de transmission sur l'une des vitesses standard.

Par défaut nous prendrons la plus courante : 9 600 baud, c'est-à-dire 9 600 bits par seconde. On peut aussi régler de manière cohérente des deux côtés une fonction optionnelle qui est le bit de parité, on choisit si on veut qu'il soit pair ou impair. Ce dernier sert à détecter une erreur sur un groupe de 8 bits. En effet, si on choisit un bit de parité pair, cela signifie que le bit de parité sera à 1 si le nombre de 1 dans les 8 bits transmis est impair, et 0 sinon. Ainsi, le nombre total de 1 dans les 8 bits transmis est le bit de parité est toujours pair.

Attention, le Raspberry Pi ne supporte que des niveaux à 3,3 Volts maximum. Cela ne devrait pas nous poser problème puisque la STM32 envoie aussi en 3,3 Volts.

## 10 Transmettre la musique par voie digitale : la MIDI - Pablo D.

Afin de comprendre comment relier notre prototype à un VST (virtual Studio Technology, synthétiseur virtuel) afin de le contrôler, nous devions comprendre comment transmettre la musique par voie digitale. J'ai ainsi mené des recherches sur cette interface.

### 10.1 Objectif du MIDI (Musical Instrument Digital Interface)

Créer un langage commun à tous les instruments électroniques de différentes marques afin qu'ils puissent communiquer entre eux.

### 10.2 Comment fonctionne le MIDI ?

Aucun signal audio (sons) n'est envoyé via MIDI. Au lieu de cela, le MIDI fonctionne comme un signal numérique. Une série de chiffres binaires (0 et 1).

Chaque instrument (ou ordinateur) comprend et répond à ces 1 et 0, qui sont combinés en messages 8 bits prenant en charge des débits de données allant jusqu'à 31 250 bits par seconde.

Ces messages peuvent communiquer des informations utiles telles que :

- Quelle note est enfoncee et le moment où une note est enfoncee et relâchée
- La vélocité (la force avec laquelle on appuie)
- L'after-touch (lorsque la pression des touches change)
- Le vibrato
- Le pitch bend

Le protocole MIDI, comme on l'appelle, peut prendre en charge jusqu'à 128 notes, allant du do cinq octaves en dessous du do moyen jusqu'au sol dix octaves au-dessus. Donc à peu près n'importe quelle note que vous pourriez souhaiter jouer.

### 10.3 Comprendre comment on règle les différents paramètres d'une note avec des 1 et 0: un premier message "note ON"

L'octet d'état pour "Note ON" sur le canal 1 ressemble à ceci:

Bit	7	6	5	4	3	2	1	0
Meaning	COMMAND	CHANNEL						
Digit	1	0	0	1	0	0	0	1
Hex nybble	9				1			

Nous préférons le format hexadécimal au format décimal lorsque nous traitons ces octets, car chaque nybble (demi-octet, soit 4 bites) se convertit directement en un seul "chiffre" hexadécimal. Cet octet d'état a une valeur de 91 en hexadécimal.

Alors, quelle note voulons-nous qu'il joue ?

Cela est codé dans le premier bit de nos octets d'état pour cette commande, le numéro de clé.

En MIDI, le do médian correspond au numéro de touche 60 (en décimal ou 3C en hexadécimal). Le numéro de clé peut être compris entre 0 et 127, mais restons-en au do du milieu. Le deuxième octet d'état indique à

l'instrument à quel point il doit jouer fort, et s'appelle la vélocité . Cela peut être quelque peu déroutant, car nous considérons généralement la vélocité comme une mesure de la vitesse plutôt que du volume. La raison pour laquelle MIDI utilise le mot vélocité de cette manière, est que c'est la vitesse à laquelle la touche est frappée qui influence le volume.

## 10.4 Objectif suivant: comment contrôler le VST ?

Maintenant, il s'agit de trouver le moyen optimal afin de connecter notre prototype au VST Massive et d'y choisir le nombre de paramètres (et lesquels) qui seront modifiés selon la trajectoire décrite par l'utilisateur sur le TouchPAd.

L'objectif étant de pouvoir aboutir à une certaine configuration de ces paramètres suite à la trajectoire décrite sur le TouchPad. Par exemple, selon l'intensité de pression, le cutoff pourrait avoir une valeur plus élevée. Selon la longueur à laquelle l'utilisateur pose son doigt sur le Pad, le decay pourrait varier etc.



## 10.5 Mise en place de la connectique entre l'IA et le VST via UDP

Afin d'établir la connectique entre le module et le VST, nous avons choisi l'option de l'UDP networking.

Il y a de nombreuses raisons de mettre en place un réseau informatique dans les patcheurs Max. Souvent, dans les grands projets, le travail à faire dépasse ce qu'une seule machine peut gérer. Dans ces cas, il peut être utile d'utiliser un réseau existant pour "récolter" les cycles CPU des machines inutilisées, ou pour envoyer des messages de synchronisation entre les machines qui exécutent des sous-sections du patch de performance complet. En outre, les protocoles de réseau peuvent être utiles pour partager des informations entre des ordinateurs utilisés par plusieurs artistes, même dans des lieux différents. Les messages udpsend et udpreceive intègrent le protocole réseau UDP standard dans une paire d'objets Max, offrant ainsi un moyen facile de transmettre des messages entre deux ordinateurs sur un réseau. Ici, ce sera entre la Raspberry et l'ordinateur faisant tourner le VST que nous échangerons les octets MIDI afin de contrôler le VST à distance.

La manipulation des messages MIDI sur le réseau utilise les objets udpsend et udpreceive, tirant parti du protocole UDP simple et léger pour transmettre de manière flexible tout type de message ou de liste. Cela nous permettra de moduler les paramètres suivants : Cutoff, Enveloppe ADSR, Effets etc.

## 11 L'avenir du projet

Notre projet n'est pas terminé, il nous reste certaines choses à faire et d'autres à améliorer. Ce qu'il nous reste à faire à court terme :

- Créer une base de données pour entraîner notre IA. On pourrait l'obtenir en faisant participer des musiciens, professionnels de la musique, mais aussi des personnes qui n'ont jamais fait de musique dans le but que notre instrument final puisse convenir à un maximum de personnes et de profils différents. Pour créer cette dataset, on pourrait demander à ces personnes de faire des gestes sur notre surface sensible correspondant, dans un premier temps aux 6 émotions fondamentales.
- Entraîner notre modèle avec cette base de données et vérifier son bon fonctionnement.

Concernant les améliorations des fonctionnalités actuelles de l'instrument qui constituent des objectifs à moyen terme, il nous reste à :

- Trouver un meilleur procédé pour fabriquer la matrice de capteurs afin d'avoir des résultats plus fiables (utilisation de scotch, soudure thermique entre le velostat et le cuivre etc.).
- Intégrer les capteurs piézoélectriques afin d'avoir plus de données sur la dynamique des appuis sur le contrôleur de MIDI.
- Modifier le PCB en un shield pour la STM 32 afin de pouvoir augmenter les débits de communication des liaisons séries OU intégration d'un microcontrôleur sur le PCB.
- Reconsidérer la dureté du silicone, sa couleur et la texture de sa surface supérieure. Trouver un procédé de moulage évitant les bulles.
- Rendre la connexion entre l'IA et le VST filaire grâce à un câble USB (A-mâle vers B-mâle)

Et pour finir, en ce qui concerne les améliorations des fonctions de notre instrument, donc dans les objectifs à long terme, il nous reste à :

- Un objectif que nous savions être trop ambitieux pour ces quelques mois de projet mais qui améliorerait grandement l'intérêt de notre produit serait de faire en sorte que grâce à la même interface que nous avons créée, une fois le VST (virtual studio technology) réglé via l'IA, on puisse changer de mode et donc jouer avec le VST ainsi réglé.
- Cela implique de penser au design de notre instrument et d'y ajouter des boutons, en quantité minimale, notamment pour l'extinction de l'instrument et le changement de mode (réglage du VST et utilisation du VST).