Partie algorithmique:

1-La reconnaissance vocale:

Le M5StickC plus est basé sur l'architecture d'un ESP32, il peut gérer un réseau de neurones profonds, alimenté avec un microphone intégré. Cela est pratique pour notre projet de reconnaissance vocale. Il existe différents tutoriels sur la façon de former et d'exécuter un modèle de commandes vocales sur un ESP32. Mais la formation doit se faire sur une machine puissante, ce qui peut être un frein technique et aussi une durée d'apprentissage importante. On a donc décidé de le faire en deux parties. La formation du modèle est divisée en deux parties : Une formulation du modèle de base puis la formation du modèle personnalisé.

1-1 Formation du modèle de données de base :

Pour la reconnaissance vocale on dispose d'un modèle de base qui est constitué du jeu de données des commandes vocales de Google et sert d'extracteur de caractéristiques pour le modèle personnalisé. Ici les données de reconnaissances vocale de Google comprennent 65 000 énoncés d'une seconde de 30 mots courts, prononcés par des milliers de personnes différentes. Ces énoncés ont été fournis par des membres du public par l'intermédiaire du site Web AIY.

1–2 Formation du modèle de données personnalisé :

Il sera formé avec la bibliothèque TensorFlow dans le navigateur web.Tensor Flow.js est une bibliothèque open source développée par l'équipe Google Brain pour exécuter des réseaux de neurones d'apprentissage en profondeur dans l'environnement du navigateur ou du nœud. Cette Bibliothèque est basée sur des structures de données appelées tenseurs. Ils sont une généralisation de vecteurs et de matrices à des dimensions potentiellement plus élevées.

Le choix d’utiliser cette technique de découpage en deux parties est dû au fait qu’il faut beaucoup moins d'échantillons pour former un modèle personnalisé qu'un modèle de base. aussi on peut obtenir une assez bonne reconnaissance avec aussi peu que 50 échantillons.

1-3 Interface d'entraînement:

Comme l’apprentissage se fera dans le navigateur web, Une interface web doit être créée. Cette dernière à pour but d’enregistrer 20 échantillons d’une seconde représentant le nom à reconnaître et ceci en accédant au microphone du navigateur web d'où on évitera de stocker les échantillons sur notre machine en local et les charger après mais les enregistrer temporairement dans le navigateur web ce qui nous donne une meilleure performance. Une étiquette (Nom) sera associée pour chaque échantillon du nom, Des échantillons dit négatifs représentant les bruits d’une seconde et en leur affectant une étiquette (Bruit) à chacun. En chargeant le modèle de base, tensorflow permet de concevoir le modèle personnalisé qui peut être exporté dans un format adapté au M5Stick C plus.
Des boutons associés à chaque étape seront dans l’interface de l'entraînement notamment enregistrer un échantillon, entraîner le model , et télécharger le modèle entraîné.

1-3-1 Réseau de neurones

Le réseau de neurones de Tensoflow a pour but de nous donner en sortie la probabilité d’un échantillon de nom afin qu’il puisse être identique à celui prononcé. Sachant que chaque sortie d'une couche est l'entrée de la couche suivante. Chaque élément de séquence entrée de notre réseau est alors la concaténation de façon aléatoire de 2 de deux échantillons Nom_et Bruit enregistrés précédemment

La problématique pour cet apprentissage est que nous avons une classification à une seule étiquette (qui est soit Nom ou Bruit) et à plusieurs classes, donc les étiquettes sont mutuellement exclusives pour chaque donnée, ce qui signifie que chaque entrée de données ne peut

appartenir qu'à une seule classe de sortie. Ce problème traitant des sorties multi-classes implique d'utiliser un classificateur prédisant ainsi la probabilité d'une classe de sortie.

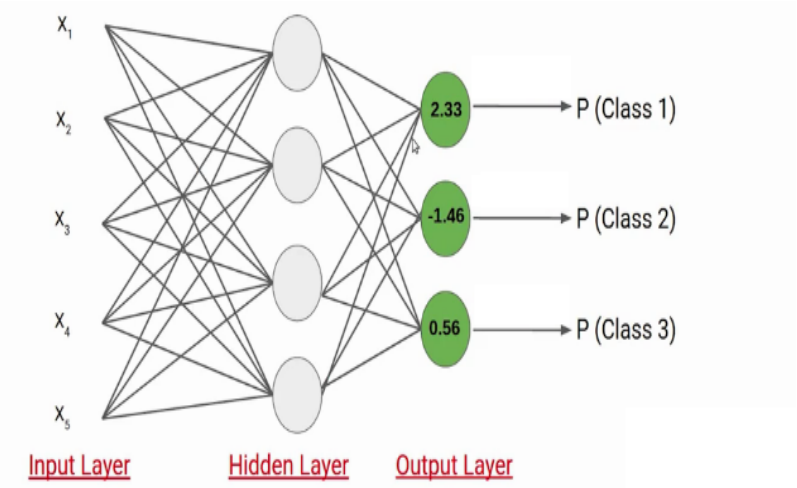

figure : Reseau de neurone Tensortflow

-La vibration:

Le modèle personnalisé généré par l'interface précédente est chargé dynamiquement en tant que module Python dans l' MCU du M5StickC plus. On utilisera le protocole I2S du microphone du M5StickC PLUS activé de façon continue pour récupérer le nom prononcé. On contrôle le vibreur connecté au M5StickC grâce à la prédiction du nom sur la base de la probabilité élevée.

Partie Réalisation:

1- Interface d'apprentissage:

1-2 Enregistrement des échantillons : l'enregistrement des échantillons se fait avec la fonction addSample() qu'on définie cette derniere crée un bouton permettant d'activer le microphone du navigateur pour enregistrer un audio. Limiter l' audio à une seconde et création de l'échantillon puis l'ajout de ce dernier dans une liste

de 20 échantillons. Une fois la liste est contient les 20 échantillons, cette fonction autorise l'ajout de l'étiquette à chaque échantillon.

1-3 Inclusion de la Bibliothèque TensorFlow:

L'interface d'apprentissage est un fichier html mais la problématique est comment ajouter TensorFlow.js à notre projet. Plusieurs méthodes existent pour obtenir TensorFlow.js .Dans notre projet pour des raisons de performances on a opté pour la version de jsDelivr qui offre le service de bibliothèques hébergées en ligne. On ajoute la balise suivante dans notre fichier html principal.

(on a opté pour la version la plus récente de tensorflow) :

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js" ></script>

On peut aussi utiliser la commande npm et installer en local TensorFlow mais due aux caractéristiques de la machine et même à la version TensorFlow existante l'apprentissage prend trop et plus de temps et peut durer plus de 24h.

1-4 Chargement du modèle de donnée de base

Les données de reconnaissance vocale de Google sont préparées dans un fichier JSON. leur chargement se fait avec la fonction prédéfinie de TensorFlow loadLayersModel()

1-5 La mise à jour des prédictions

Sur la base des mots du modèle de base il faut définir une prédiction qui se met à jour instantanément en comparant l'échantillon nom au vocal correspondant à chaque mot du du modèle de base .

1-6 L'apprentissage propre

On définit la fonction trainModel() qui permet de charger des échantillons de nom et bruit pour les donner en entrée a tensorflow. Puis elle concaté des 2 types d'échantillons la fonction propre à tensorflow sequentiel() crée les couches de notre réseau de neurone dans lequel chaque sortie d'une couche est l'entrée de la couche suivante. En termes simples, nous pouvons dire qu'il s'agit d'un empilement linéaire de couches sans ramification ni saut. Dans la dernière couche la fonction softmax() de TesorFlow prédit ainsi la probabilité d'une classe sur la base de la régression logistique utilisée pour les problèmes de classification, qui pour une entrée, renvoie un nombre réel entre 0 et 1 pour chaque classe. Une fois l'apprentissage est fait dans la même fonction on initialise les paramètres NNom avec la fonction initQuantizedModel() qu'on définit dans la partie suivante.

1-7 Réseaux de neurones pour M5Stick C plus

Comme notre apprentissage doit être déploré dans un M5Stick C plus (zone rouge de la figure ci-après), ce dernier doit être adapté à l'architecture du microcontrôleur ESP32 du M5Stick C plus. On définit une fonction initQuantizedModel() qui initialise les paramètres de la bibliothèque de réseaux neurones nommée NNOM dédiée à ce type d'architectures.

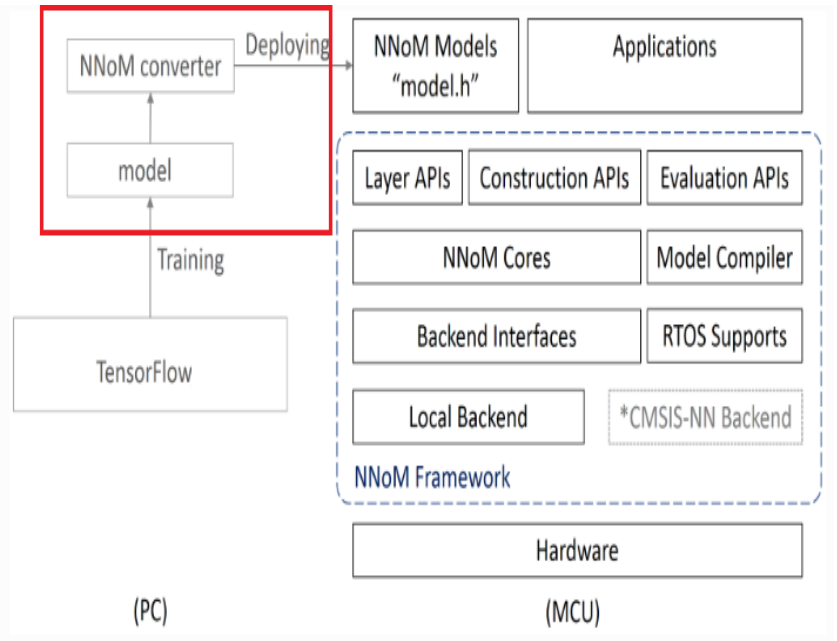

figure - Deploration TensorFlow NNOM

1-8 Exportation du modèle entraîné:

On définit une fonction downloadModel() qui définit un bouton permettant de télécharger le modèle entraîné sous format d'un fichier python. La conversion du modèle dans un format ASCII interprétable par le M5stick C plus se fait avec la fonction propre de python. a2b_base64() qui convertit un bloc de données en base64 en binaire. le fichier python exporté doit contenir une fonction predict() qu'on définie et qui a pour rôle de prévoir la probabilité d'un audio récupéré par le micro du sous format d'un pourcentage interprétable et comparable à une valeur entière.

2-La vibration:

Une fois on a exporté le modèle entraîné (le fichier data.py) on ecrit ce dernier grace à la commande %writefile

On crée un script mycropython et dans la boucle while principale infinie :

1. On récupère avec la fonction readinfo() propre au microphone les paroles sous forme de buffer (un tableau de type bytearray)
2. On prédit la probabilité grâce à la fonction predict() du modèle de données déjà exporté et écrite dans le M5StickC Plus.
3. Si la probabilité est < 70% ou si l'etiquette détecté est un Bruit donc on ignore et on continue la lecture des données du microphone.
4. Si on détecte le label NOM donc le nom est reconnu on active le vibreur avec la classe Pin pour initialiser le pin 25 (figure ci-après) du M5StickC plus en tant que sortie et nous utilisons la méthode value(1) pour déclencher la vibration et la méthode value(0) pour l'arrêter. Avec la fonction sleep() on fait une pause pour fixer la durée de la vibration (qui est de 0.5 secondes pour notre cas).

Figure : Montage des pins vibreur et M5Stick c plus.

Liens:

1-code mycropython pour controller le vibreur
https://github.com/lougani-faouzi/Speech-recongnition/blob/main/write_model.py
2- modèle entraîné exporté après apprentissage
https://github.com/lougani-faouzi/Speech-recongnition/blob/main/data.py
3-interface d'apprentissage
https://github.com/lougani-faouzi/Speech-recongnition/blob/main/speech-commands.html
4- vidéo test de reconnaissance d'un mot LILI
https://drive.google.com/file/d/1RrexvhuW6xALLlbQ22HcjvxcHPNbWMzF/view?usp=sharing