l'œil qui réfléchit

reconnaissance faciale et plus encore, utilisant l'ESP32-S3-EYE

Tam Hanna (Hongrie)

La carte ESP32-S3-EYE d'Espressif est une plateforme spécialement conçue pour évaluer les applications de reconnaissance faciale et autres applications d'Intelligence Artificielle. Elle est fournie avec un écosystème de logiciels et de nombreux exemples, provenant de l'ensemble d'outils ESP-WHO du fabricant en ce qui concerne la reconnaissance faciale, et de TensorFlow pour les opérations manuelles. Allons les découvrir !

Les progrès et la sophistication grandissante des algorithmes d'Intelligence Artificielle (IA) ont été fulgurants ces dernières années. Il existe maintenant des systèmes très performants qui fournissent des résultats impressionnants, comparables à ceux d'un être humain (en particulier dans le cas de l'IA hébergée dans le nuage). La loi de Moore, et l'accroissement constant de la demande des utilisateurs, imposent aux fabricants d'inclure des composants optimisés pour l'IA dans leur offre de dispositifs. En ce qui concerne Espressif, il s'agit de l'ESP32 dans sa variante S3 qui, grâce à ses instructions vectorisées, offre toutes sortes de moteurs d'accélération d'Intelligence Artificielle et un ensemble spécifique d'instructions dédiées à l'IA.
Ces derniers mois, un écosystème robuste et évolutif s'est développé autour de l'ESP32-S3, facilitant la création de diverses applications d'IA par les développeurs. Dans cet article, nous allons détailler et expérimenter les possibilités de cette technologie.

Débuts sans appréhension

Espressif est conscient des besoins des développeurs. Depuis le lancement de la carte ESP32-LyraT, prévue pour les applications audio-phoniques, il y a quelques années, le fabricant à constamment introduit des cartes d'évaluation pour des «besoins spécifiques». Nous allons utiliser l'ESP32-S3-Eye pour exécuter l'ensemble des exemples décrits dans les paragraphes suivants. La **figure 1** est un schéma des blocs fonctionnels des composants de la carte à circuit-imprimé. La présence

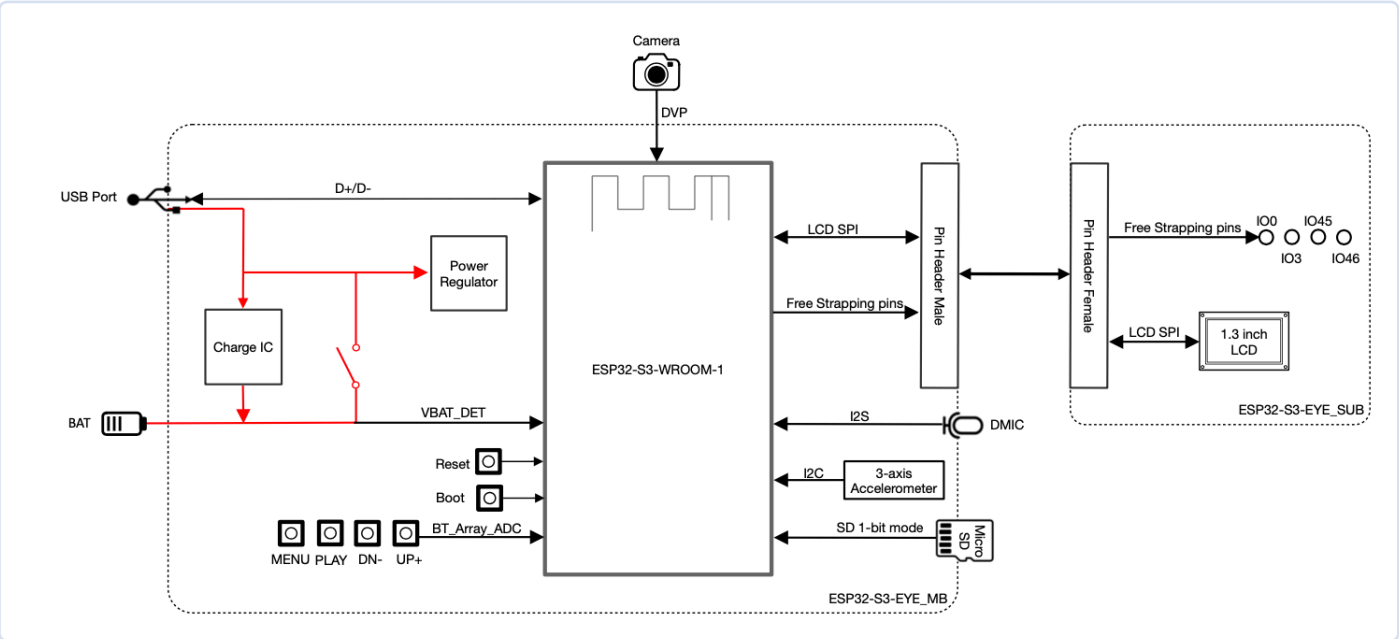


Figure 1. Blocs fonctionnels de l'ESP32-S3-EYE (Source : [10]).

sur la carte d'une caméra et d'un petit écran LCD [1] m'a convaincu d'utiliser cette carte d'évaluation dont le prix est d'environ 42 €. Cet écran couleurs se révèle être très utile, permettant de voir directement les résultats du processus d'apprentissage de ce dispositif, sans nécessiter l'utilisation d'un PC. Un microphone numérique est situé à la surface du circuit-imprimé, sous l'écran. Il peut être utilisé pour tester divers logiciels de reconnaissance du langage. Il est important de noter que les expérimentations entreprises peuvent également être faites en utilisant d'autres plateformes matérielles. Le schéma de l'ESP32-S3-Eye se trouve en [2], il peut servir de base pour la réalisation de circuits personnalisés. Les composants utilisés par Espressif sont en général disponibles chez plusieurs distributeurs.

Disponibilité des composants ES8388

Se procurer les composants recommandés par Espressif nécessite parfois une approche différente. En ce qui concerne les codecs audio ES8388, je n'ai pas réussi à trouver un distributeur, basé aux USA ou en Europe, qui ne l'ait en stock. Une demande directe à son fabricant m'a toutefois permis d'être orienté vers Newtech Component Ltd, un distributeur qui a été suffisamment coopératif pour m'en procurer gratuitement 30 exemplaires. Pour cela, il convient de prévoir un transporteur pouvant expédier les échantillons à une adresse locale. Pour cela, l'auteur utilise TipTrans.

La carte est fournie avec le micrologiciel installé ce qui évite tout problème durant la procédure de mise en œuvre du système. Si la carte ESP32-S3-EYE est à l'état usine, vous n'avez qu'à la mettre sous tension par le port micro-USB et attendre quelques secondes. L'application de reconnaissance faciale apparaîtra ensuite, comme illustré sur la **figure 2**.
Il est important de remarquer qu'à cette étape, les algorithmes d'IA et d'apprentissage (ML ou Machine Learning), font preuve d'une excellente résilience relativement à la qualité des données capturées. La photographie de la figure 2 a été prise en conservant le film de protection de l'objectif de la caméra. La réduction de contraste qui en résulte (et malgré que je ne sois pas rasé), n'a pas pris en défaut le processus d'IA de reconnaissance faciale.

Retour vers le futur

Parfois, vous souhaiterez peut-être réinitialiser votre ESP32-S3-EYE à «l'état usine». Les fichiers .bin précompilés se trouvent à l'adresse https://github.com/espressif/esp-who/tree/master/default_bin. Ils peuvent être "flashés" sur la carte de façon identique à tout autre fichier binaire.

Première expérimentation

La présence, quasiment illimitée, de capital-risque, a conduit à la création d'une multitude de plateformes d'IA. La logique impose que ces sociétés ne considèrent pas uniquement les PC, mais incluent également des microcontrôleurs dans leurs listes de plateformes supportées. Une revue détaillée de l'état actuel du marché se situerait bien au-delà

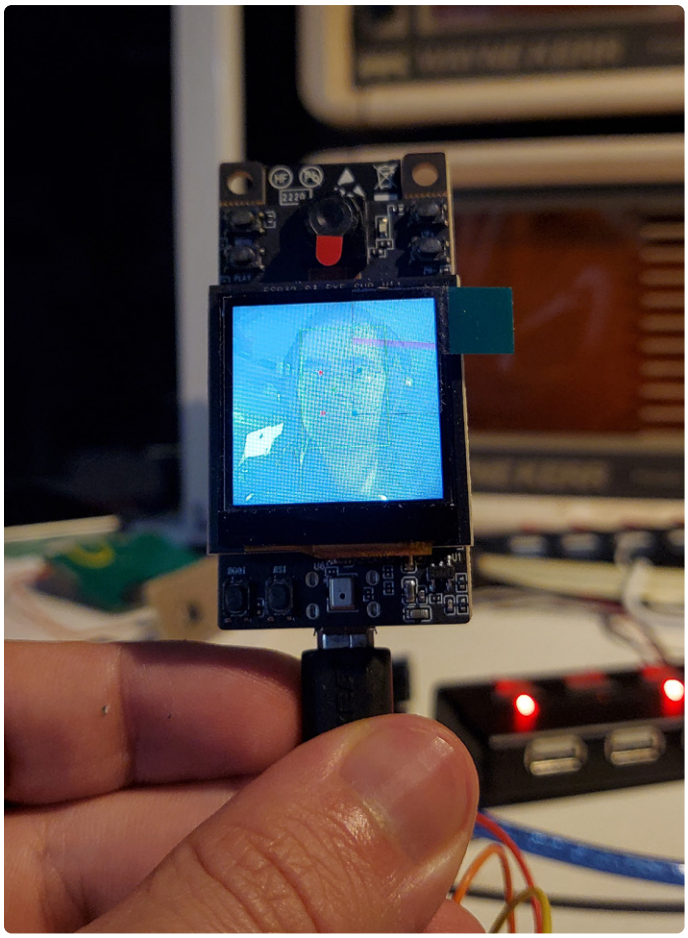


Figure 2. Bien que mon laboratoire soit sombre, la caméra fonctionne malgré la présence du film de protection de l'objectif !

des objectifs de cet article, c'est pourquoi, pour cette première étape, nous nous limiterons à expérimenter le processus de développement du traitement d'images sur la plateforme *ESP-WHO* proposée par Espressif. Il s'agit d'une plateforme «optimisée» pour certains types de reconnaissance d'images animées. Son diagramme structurel est illustré **figure 3**.

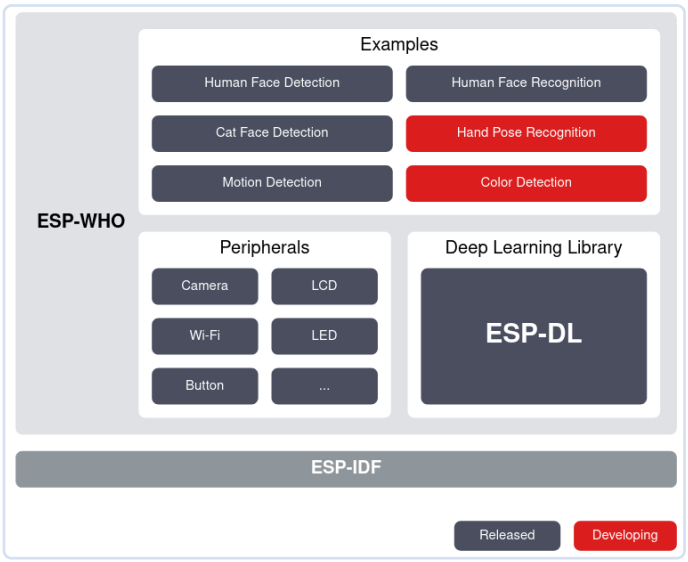


Figure 3. L'ESP-WHO utilise quelques autres composantes de l'écosystème Espressif (Source : [11]).

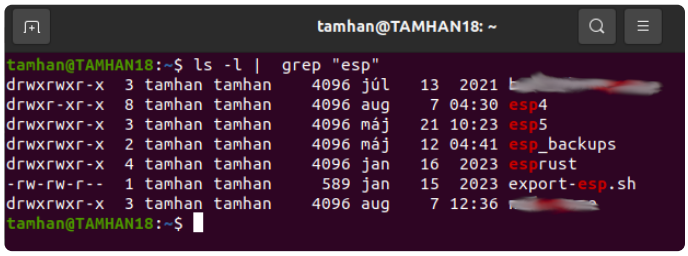


Figure 4. Différentes versions de l'ESP-IDF peuvent en général coexister sur la même station de travail.

La bibliothèque *ESP Deep Learning Library* (ESP-DL), disponible en [3], est une ressource importante. Il s'agit essentiellement d'un ensemble optimisé offrant diverses techniques d'IA et réduisant la latence lorsqu'on l'utilise conjointement avec un accélérateur matériel. Il convient de noter que, tels que présents dans l'ensemble, les composants WHO fonctionnent sous la version 4.4 de l'*ESP-IDF* et ne permettent pas encore un fonctionnement sous la version 5.0. Pour les projets de mes activités de consulting, j'utilise généralement la version 4.4, mais je dispose également de plusieurs variantes d'environnements de développement ESP32, installés sur mon ordinateur comme le montre le **figure 4**.
La version IDF utilisée dans les étapes suivantes, est identifiée ainsi :

```
tamhan@TAMHAN18:~/esp4/esp-idf$ idf.py --version
ESP-IDF v4.4.4-dirty
```

Nous sommes maintenant prêts pour le téléchargement du code *ESP-WHO.* La commande `git` suivante le permet :

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://
github.com/espressif/esp-who.git
. . .
```

L'exécution de la commande de compression d'objets `Compressing objects:` peut nécessiter un temps d'attente important, les dépôts sont parfois assez volumineux et lents à manipuler. Après avoir terminé cette étape, il est recommandé de procéder à l'initialisation du sous-ensemble, selon le processus suivant :

```
tamhan@TAMHAN18:~/esp4$ cd esp-who/
tamhan@TAMHAN18:~/esp4/esp-who$ git submodule update
--recursive --init
```

Dans la plupart des cas, la commande `git submodule update --recursive --init` ne produira aucun affichage à l'écran. Cela signifie que l'image en cours de traitement est « complète ».

Quelques exemples

Le logiciel de reconnaissance faciale *ESP-WHO* fourni par Espressif comporte trois options d'interface. Des exemples de projets sont présents sur la **figure 6**.

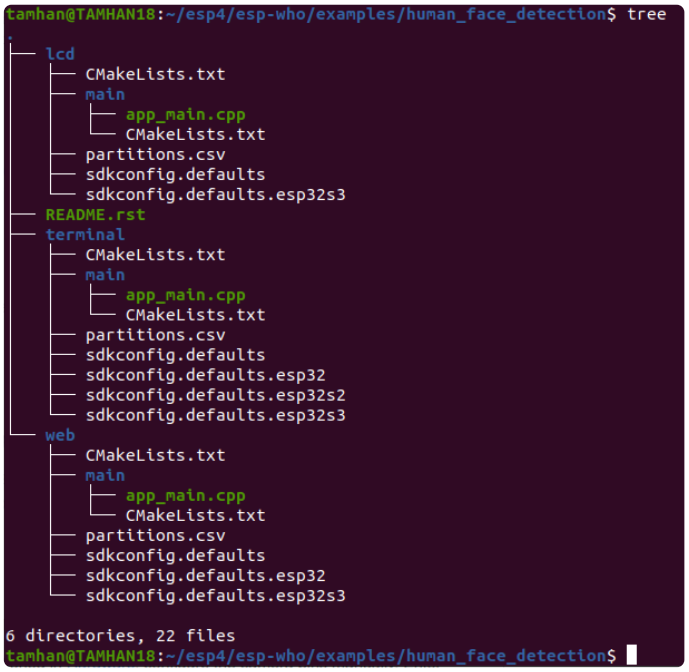


Figure 5. Variantes ESP-IDF dont je dispose !

Dans les étapes qui suivent, je vais utiliser la variante lcd des exemples `~/esp4/esp-who/examples/human_face_detection`. Les résultats obtenus seront donc directement affichés sur le petit écran présent sur la carte ESP32. La variante *Terminal* utilise le moniteur de communication idf-py, alors que la variante *Web* met en jeu un serveur web. Pour commencer, vous devez spécifier le type de contrôleur ESP32 que vous souhaitez utiliser selon la procédure suivante. Si, comme moi, vous utilisez une carte ESP32-S3-EYE, la chaine `esp32s3` est transmise, par la commande :

```
tamhan@TAMHAN18:~/esp4/esp-who/examples/human_face_detec-
tion/lcd$ idf.py set-target esp32s3
```

Le code source actuel de l'exemple est d'une simplicité impressionnante, il se trouve dans le fichier *main/app_main.cpp*. Pour en avoir une vue globale meilleure, je vais tout d'abord l'imprimer complètement, avant d'y ajouter des commentaires (voir le **listage 1)**.
Le cœur du moteur ESP-WHO utilise l'objet du noyau *queue* présent dans FreeRTOS, décrit en détail en [4]. Les deux déclarations statiques créent une file d'attente (queue) d'entrée et sortie qui sera ensuite utilisée pour manipule les données via le pipeline de reconnaissance (recognition pipeline).
Le reste du code, contribution du développeur, se consacre principalement à la mise en œuvre d'un pipeline par l'appel à trois fonctions. Les analogies au pipeline utilisé dans l'ESP-ADF la plateforme audio [5] sont évidentes.
Pour continuer la mise en œuvre, vous devrez entrer la commande `idf.py menuconfig` dans la première étape, afin de charger l'environnement correspondant à la configuration habituelle *menuconfig,*

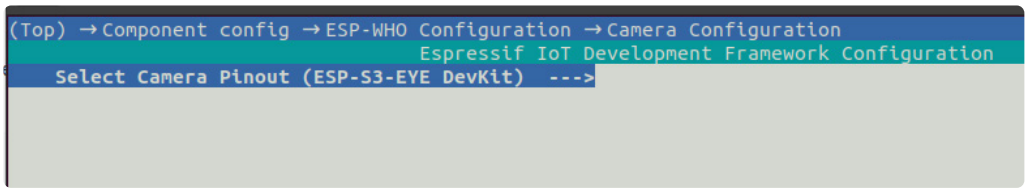


Figure 6. Si vous utilisez un module caméra existant, vous devez le reconfigurer ici !

Listage 1. Code source de l'exemple.

```
#include "who_camera.h"
#include "who_human_face_detection.hpp"
#include "who_lcd.h"

static QueueHandle_t xQueueAIFrame = NULL;
static QueueHandle_t xQueueLCDFrame = NULL;

extern "C" void app_main()
{
    xQueueAIFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueLCDFrame = xQueueCreate(2, sizeof(camera_fb_t *));

    register_camera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2, xQueueAIFrame);
    register_human_face_detection(xQueueAIFrame, NULL, NULL, xQueueLCDFrame, false);
    register_lcd(xQueueLCDFrame, NULL, true);
}
```

utilisée dans les autres projets ESP32 (ainsi que dans le noyau Linux). Rendez-vous ensuite, dans la section *Component config → ESP-WHO Configuration*. Dans le champ *Camera Configuration,* assurez-vous que la caméra de votre carte d'évaluation est préconfigurée comme le montre la **figure 6.**

Enregistrez la configuration créée dans menuconfig, puis exécutez la commande habituelle `idf.py build` pour lancer la compilation. La création de l'image exécutable prendra un peu plus de temps, car approximativement 1200 fichiers doivent être compilés par la plate-forme de développement.

Avant de charger le logiciel dans la carte par la commande `idf.py flash` suivie du numéro de port, vous devrez mettre la carte en mode bootloader. Utilisez pour cela les deux boutons situés à proximité du connecteur USB. Maintenez appuyé le bouton BOOT tout en appuyant momentanément sur le bouton RST, relâchez ensuite les deux boutons. Vous pouvez maintenant continuer par le chargement du programme dans l'ESP32, selon la procédure habituelle. Vous pourrez vérifier la réussite du chargement du mode bootloader dans *dmesg*, comme l'indique la **figure 7**. Appuyez à nouveau sur le bouton RST afin de lancer l'exécution de la nouvelle variante de la reconnaissance faciale familiale (figure 2).

Examen succinct du code

L'utilisation de l'ESP-WHO est une introduction relativement simple destinée aux développeurs qui commencent à expérimenter des applications d'Intelligence Artificielle, sans devoir fournir un effort important. Espressif fournit également le code source de certains composants [6], permettant d'avoir une connaissance des routines de reconnaissance faciale.

Il utilise le détecteur préexistant `HumanFaceDetectMSR01`, qui reçoit certains paramètres transmis lors de la procédure de paramétrisation :

```
static void task_process_handler(void *arg) {
  camera_fb_t *frame = NULL;
  HumanFaceDetectMSR01 detector
      (0.3F, 0.3F, 10, 0.3F);
```

Le travail réel est effectué par le modèle ML (apprentissage machine) dans une boucle infinie qui reçoit les images individuelles à traiter de la queue créée préalablement :

```
while (true) {
   bool is_detected = false;
   if (xQueueReceive(xQueueFrameI,
     &frame, portMAX_DELAY)) {
     std::list<dl::detect::result_t> &detect_results =
       detector.infer((uint16_t *)frame->buf,
      {(int)frame->height, (int)frame->width, 3});
```

Les appels à la fonction `detector.infer()` permettent de s'assurer que le process d'inférence est exécuté. Si l'objet retourné `results` comporte des résultats de détection, le programme fait appel à deux fonctions de sortie :

```
    if (detect_results.size() > 0) {
     draw_detection_result((uint16_t *)frame->buf,
     frame->height, frame->width, detect_results);
     print_detection_result(detect_results);
     is_detected = true;
    }
   }
```

```
[ 8792.171038] usb 1-1.5: new full-speed USB device number 6 using ehci-pci
[ 8792.301169] usb 1-1.5: New USB device found, idVendor=303a, idProduct=1001, bcdDevice= 1.01
[ 8792.301175] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 8792.301178] usb 1-1.5: Product: USB JTAG/serial debug unit
[ 8792.301181] usb 1-1.5: Manufacturer: Espressif
[ 8792.301183] usb 1-1.5: SerialNumber: 34:85:18:8C:50:D8
[ 8792.301642] cdc_acm 1-1.5:1.0: ttyACM0: USB ACM device
tamhan@TAMHAN18:~$
```

Figure 7. Ce message d'état indique une connexion réussie.

Performance Comparison

A quick summary of ESP-NN optimisations, measured on various chipsets:

Target	TFLite Micro Example	without ESP-NN	with ESP-NN	CPU Freq
ESP32-S3	Person Detection	2300ms	54ms	240MHz
ESP32	Person Detection	4084ms	380ms	240MHz
ESP32-C3	Person Detection	3355ms	426ms	160MHz

Figure 8. L'activation de l'accélérateur d'IA permet d'obtenir un résultat plus rapidement. (Source : [9]).

Le reste de l'ESP-WHO est généralement constitué de composants logiciels préexistants, comme par exemple ESP-Cam [7], permettant l'accès à la caméra. Je n'irai pas plus loin dans le logiciel ESP-WHO car les exemples fournis sont explicites.

Approfondissons en utilisant TensorFlow

Ceux qui souhaitent un contrôle plus approfondi du comportement de leur système seront tentés de partir de zéro pour la programmation de l'ensemble, cela n'est toutefois pas une solution pratique. Sauf si une grande attention y est apportée, le système pourrait rapidement devenir incontrôlable et conduire à des coûts élevés de maintenance du logiciel au cours de son existence.

La bibliothèque Google *TensorFlow*, disponible à [8], est devenue quasi standard et existe, depuis pas mal de temps, dans des versions optimisées, destinées à divers microcontrôleurs. Il est important de remarquer que dans GitHub, elle se trouve dans deux dépôts ; la raison en est surprenante, car Google est passé de la distribution de code générique et spécifique à un fabricant, à la bibliothèque *TensorFlow,* en cours du développement de cette plateforme. La version de cette bibliothèque se trouve en [9].

Étant donné que TensorFlow accède à diverses composantes en arrière-plan de la plateforme ESP-IDF, le déploiement à partir GitHub devra se faire en incluant le paramètre `--recursive` qui délègue à la ligne de commande la nécessité d'indiquer l'endroit où se trouve le code et de s'assurer que l'on dispose du dépôt principal et l'ensemble des sous-modules.

```
tamhan@TAMHAN18:~/esp4$ git clone --recursive https://
github.com/espressif/tflite-micro-esp-examples.git
```

Pour un test initial de mise en œuvre, l'exemple *~/esp4/tflite-micro-esp-examples/examples/hello_world* peut être utilisé. Il utilise un modèle d'apprentissage (ML) optimisé par la prédiction des valeurs d'une fonction sinusoïdale; alternative à l'algorithme conventionnel CORDIC, et d'un modèle qui fonctionne avec moins de données en entrée. Dans l'étape suivante, entrez à nouveau la commande `idf.py set-target esp32s3` afin d'optimiser le squelette de l'application pour l'ESP-32-S3.

Selon la version de l'ESP-IDF disponible sur votre station de travail ou votre PC, vous pourrez avoir des messages d'erreur lors du paramétrage du code téléchargé depuis le dépôt (*Invalid manifest...)*. Ces erreurs proviennent de composantes partiellement obsolètes dans la chaîne de compilation. Pour résoudre ce problème, lancez l'exécution du programme suivant à partir de la ligne de commande :

```
/home/tamhan/.espressif/python_env/idf4.4_py3.8_env/bin/
python -m pip install --upgrade idf-component-manager
```

Ouvrez ensuite une fenêtre *Nautilus* pointant sur le répertoire contenant le squelette du projet, en entrant la commande suivante. La suppression du répertoire de préparation de l'exécutable doit être faite manuellement de telle façon que la commande `set-target` puisse à nouveau créer les fichiers de travail de la compilation :

```
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ nautilus .
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ idf.py set-target esp32s3
tamhan@TAMHAN18:~/esp4/tflite-micro-esp-examples/examples/
hello_world$ idf.py menuconfig
```

Notez l'entrée *ESP-NN* dans *Menuconfig.* Elle permet le fonctionnement de la configuration de la bibliothèque de chargement dynamique (DL ou Dynamic Loader). La sélection de l'option *Optimized versions* dans la section *Optimization for neural network functions* est fortement recommandée car elle active l'accélérateur. Les performances indiquées sur la **figure 8** montrent l'accroissement signifiant que l'on obtient quand cette option est validée.

```
I (292) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
x_value: 0.000000, y_value: 0.000000

x_value: 0.314159, y_value: 0.372770

x_value: 0.628319, y_value: 0.559154

x_value: 0.942478, y_value: 0.838731

x_value: 1.256637, y_value: 0.965812

x_value: 1.570796, y_value: 1.042060

x_value: 1.884956, y_value: 0.957340

x_value: 2.199115, y_value: 0.821787

x_value: 2.513274, y_value: 0.533738

x_value: 2.827433, y_value: 0.237217

x_value: 3.141593, y_value: 0.008472

x_value: 3.455752, y_value: -0.304993

x_value: 3.769912, y_value: -0.533738

x_value: 4.084070, y_value: -0.779427

x_value: 4.398230, y_value: -0.965812

x_value: 4.712389, y_value: -1.109837
```

Figure 9. Un réseau neuronal fonctionne également avec des données sinusoïdales.

La compilation et l'exécution vont maintenant se dérouler comme prévu. Vous pouvez voir la sortie des résultats obtenus sur la **figure 9**.

Analyse du code TensorFlow

Ne soyez pas surpris si des images chaotiques apparaissent sur l'écran de l'ESP-EYE, l'écran LCD intégré a son propre contrôleur d'image, il ne visualise que la dernière image enregistrée jusqu'à ce qu'une nouvelle mise à jour intervienne.
Si vous visualisez le fichier main.cc dans votre éditeur de texte de votre choix, vous trouverez le fragment de code suivant :

```
#include «main_functions.h»

extern «C» void app_main(void) {
  setup();
  while (true) {
   loop();
  }
}
```

Vos premières impressions ne vous trompent pas, TensorFlow est proche de l'environnement Arduino. La définition des fonctions `setup()` et `loop()` se trouve dans le fichier *main_functions.cc*.
La constitution de la fonction `loop()` est particulièrement intéressante, car elle est responsable de l'exécution des échanges. Sa première tâche consiste à générer les données d'entrée qui seront transmises au prédicteur sinusoïdal :

```
void loop() {
  float position =
   static_cast<float>(inference_count) /
   static_cast<float>(kInferencesPerCycle);
  float x = position * kXrange;
```

À l'étape suivante, l'information est quantifiée afin de la rendre plus « digeste » par le réseau neuronal. Il s'agit d'une approche habituelle en matière d'apprentissage machine (Machine Learning). La normalisation de toutes les données d'entrée entre 0 et 1 permet de contrôler la complexité des résultats algorithmiques :

```
  int8_t x_quantized =
    x / input->params.scale +
     input->params.zero_point;
  input->data.int8[0] = x_quantized;
```

Les calculs et la quantification interviennent dans le bloc suivant :

```
  TfLiteStatus invoke_status =
      interpreter->Invoke();
  if (invoke_status != kTfLiteOk) {
   MicroPrintf(«Invoke failed on x: %f\n»,
    static_cast<double>(x));
   return;
  }
  int8_t y_quantized = output->data.int8[0];
  float y = (y_quantized -
     output->params.zero_point) *
     output->params.scale;
```

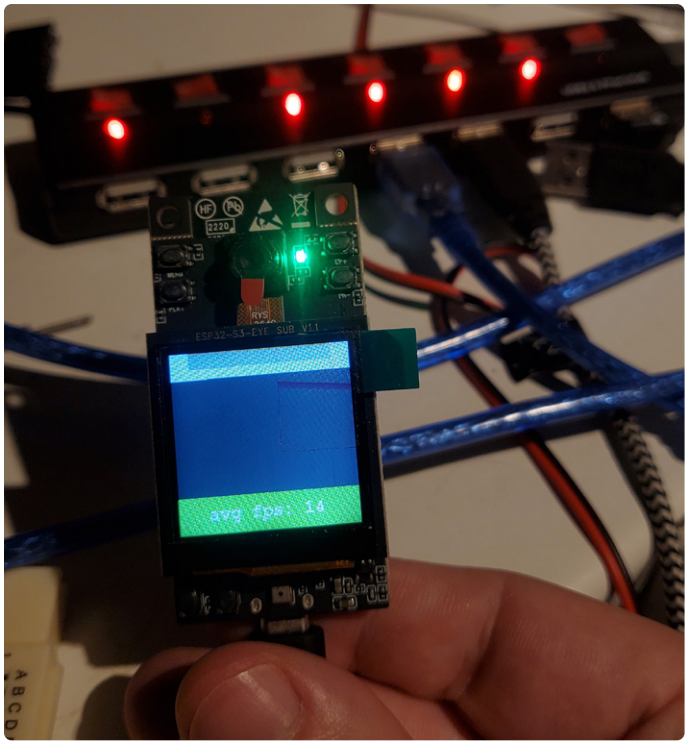


Figure 10. Le bandeau vert en bas d'écran indique une reconnaissance réussie

Pour terminer, un nettoyage final est nécessaire. Il est intéressant de noter que selon la documentation de TensorFlow, la fonction `HandleOutput()` en charge de la sortie des données, doit être fournie par le développeur :

```
  HandleOutput(x, y);
  inference_count += 1;
  if (inference_count >= kInferencesPerCycle)
      inference_count = 0;
}
}
```

Expérimentation des modules plus évolués

Si vous analysez en détail les exemples contenus dans le répertoire *~/esp4/tflite-micro-esp-examples/examples,* fourni par Google, vous constaterez qu'un exemple de reconnaissance de la parole nommé *micro_speech* y est présent, ainsi qu'un exemple de reconnaissance faciale appelé *person_detection.* Pour obtenir ces exemples et en lancer l'exécution, il est nécessaire de suivre les trois étapes habituelles des procédures de paramétrage, vérifier les paramètres de *menuconfig,* suivie du téléversement du code machine de l'ESP32.
Il est important de noter que ces exemples avancés fonctionnent en utilisant les échanges par la ligne de commande. Si vous souhaitez ajouter la possibilité de sorties sur l'écran des données du détecteur d'identité, vous devrez adapter le fichier *esp_main.h* comme il suit :

```
// Enable this to do inference on embedded images
//#define CLI_ONLY_INFERENCE 1
#define DISPLAY_SUPPORT 1
```

Après recompilation, le contenu de l'écran représenté sur la **figure 10** apparaîtra. L'étude des problèmes relatifs à l'affichage pourraient faire l'objet d'un autre article. Une reconnaissance réussie est indiquée lorsque le bas de l'écran devient vert.

Deux méthodologies

Les expérimentations menées ici démontrent que la plateforme ESP32-S3 est capable de réaliser des tâches de reconnaissance d'objets et faciale de différentes façons. Tandis que l'utilisation des possibilités offertes par l'ESP-WHO permet d'obtenir rapidement des résultats appréciables, la méthode intégrant l'écosystème TensorFlow permet d'utiliser des techniques avancées dans le domaine de l'apprentissage machine.

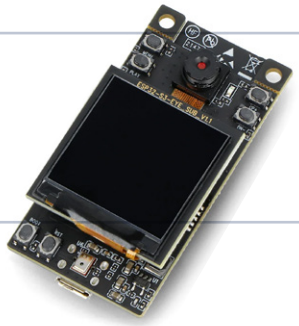
VF : Jean Boyer — 230556-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (tamhan@tamoggemon.com) ou contactez Elektor (redaction@elektor.fr).

Produit

> **ESP32-S3-EYE**
> www.elektor.fr/20626



LIENS

[1] Recherche de distributeurs de l'ESP32-S3-EYE : https://oemsecrets.com/compare/ESP32-S3-EYE
[2] Schéma de l'ESP32-S3-EYE : https://tinyurl.com/esp32eyeschematics
[3] Bibliothèque d'apprentissage profond pour l'ESP : https://github.com/espressif/esp-dl
[4] FreeRTOS Queues : https://freertos.org/a00018.html
[5] T. Hanna, « les signaux audios et l'ESP32 », Elektor 1-2/2023 : https://www.elektormagazine.fr/magazine/elektor-291/61422
[6] Exemples d'IA avec l'ESP-WHO : https://github.com/espressif/esp-who/tree/master/components/modules/ai
[7] Contrôle d'une caméra avec l'ESP32 : https://github.com/espressif/esp32-camera
[8] TensorFlow : https://tensorflow.org
[9] TensorFlow Lite Micro : https://github.com/espressif/tflite-micro-esp-examples
[10] Schéma des blocs fonctionnels de l'ESP32-S3-EYE : https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md
[11] ESP-WHO sur GitHub : https://github.com/espressif/esp-who