Het denkende oog

gezichtsherkenning en meer met de ESP32-S3-EYE

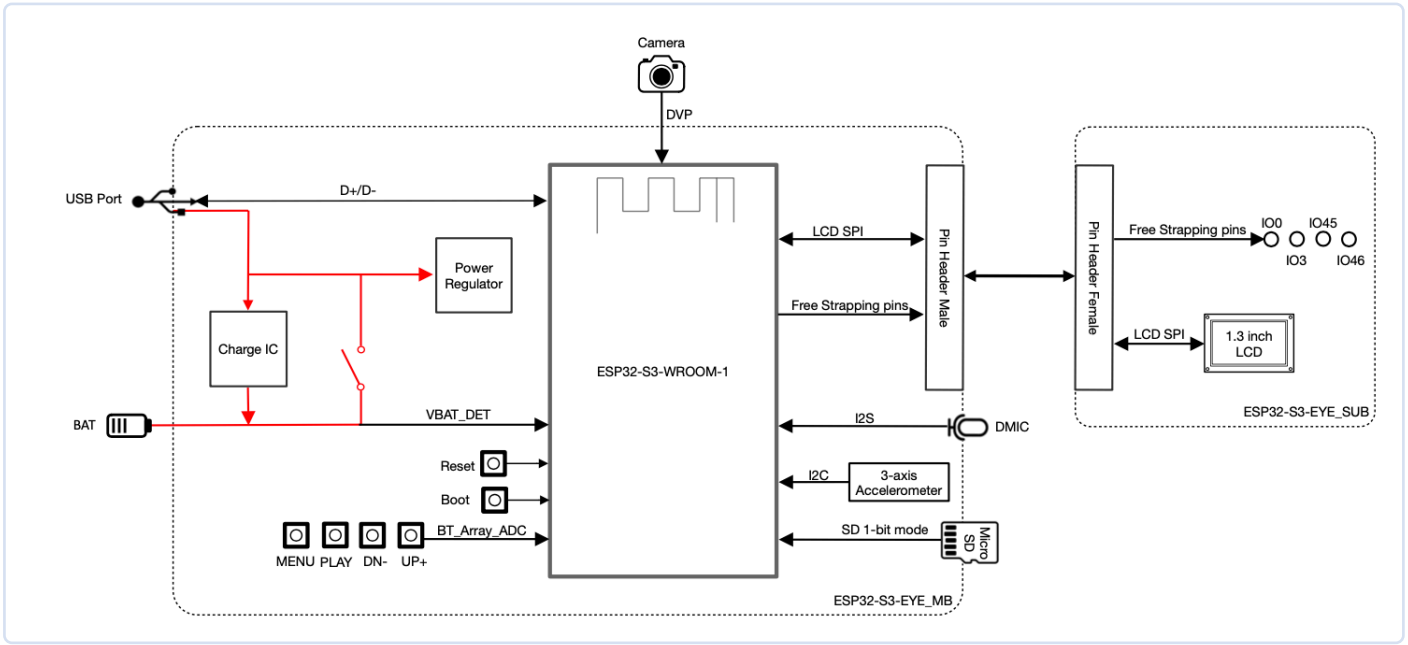
By Tam Hanna (Hongarije)

Het ESP32-S3-EYE-board van Espressif is een platform dat speciaal is ontworpen voor het evalueren van beeldherkennings- en andere AI-toepassingen. Het wordt geleverd met een software-ecosysteem en talloze voorbeelden, zowel voor het ESP-WHO framework voor gezichtsherkenning van de fabrikant als voor handmatige bediening met TensorFlow. Handen uit de mouwen dus!

Algoritmen voor kunstmatige intelligentie hebben de afgelopen jaren een enorme ontwikkeling doorgemaakt en zijn veel complexer geworden. Er zijn nu krachtige systemen die indrukwekkende prestaties leveren die lijken op die van mensen (vooral in het geval van AI die in de cloud wordt gehost). De wet van Moore en de steeds hogere eisen van eindgebruikers zorgen ervoor dat veel fabrikanten van microcontrollers 'AI-geoptimaliseerde' chips in hun assortiment opnemen. In het geval van Espressif is dit de ESP32 in de S3-variant, die met vector-instructies een soort AI-versneller of een specifieke AI-instructieset biedt. Rond deze ESP32-S3 is de afgelopen maanden een degelijk en actief ecosysteem van ontwikkelaars ontstaan, waardoor het voor gebruikers gemakkelijker is geworden om verschillende toepassingen voor kunstmatige intelligentie te implementeren. In dit artikel zullen we de mogelijkheden nader bekijken en met de technologie experimenteren.

Opstarten zonder stress

Espressif is zich terdege bewust van de behoeften van ontwikkelaars. Sinds de introductie van het ESP32-LyraT-board een paar jaar geleden, dat gericht was op audio-toepassingen, heeft het bedrijf consequent evaluatieboards geïntroduceerd die geoptimaliseerd zijn voor 'speciale' toepassingen.



Figuur 1. Het blokschema van de ESP32-S3-EYE (bron: [10]).

Ik zal de ESP32-S3-Eye gebruiken om alle voorbeelden uit te voeren die hieronder worden beschreven. **Figuur 1** toont een blokschema van alle componenten op de print.
Een belangrijke reden om dit evaluatieboard te gebruiken, dat in de VS ongeveer $ 45 kost, is de aanwezigheid van zowel een camera als een klein LC-display op de printplaat. [1] Dit kleurenscherm bleek erg nuttig te zijn, omdat de resultaten van het machine learning-proces direct zichtbaar zijn, zonder dat er een PC aan te pas hoeft te komen. Op de voorzijde van de print, net onder het scherm, is een digitale microfoon gemonteerd. Deze kan gebruikt worden om verschillende spraakherkennings-engines te testen.
Het is de moeite waard om op te merken dat de experimenten van dit artikel ook uitgevoerd zouden kunnen worden op een ander hardware-platform. Het schema van het ESP32-S3-EYE-board vind je op [2], en ik beveel het aan als uitgangspunt voor aangepaste schakelingen. De door Espressif gebruikte componenten zijn over het algemeen gemakkelijk verkrijgbaar bij veel leveranciers.

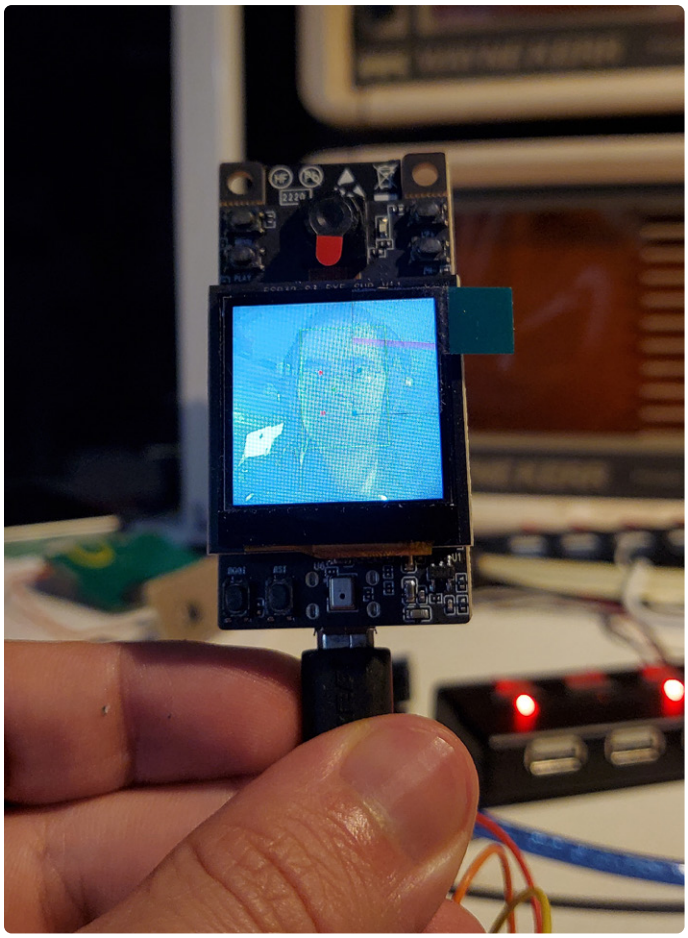
Beschikbaarheid van onderdelen en de ES8388

Het vinden van de halfgeleidercomponenten waar Espressif de voorkeur aan geeft, vereist soms een andere aanpak. In het geval van de ES8388-audiocodec kon ik geen van de traditionele distributeurs in de VS of Europa vinden die dit onderdeel op voorraad had. Een directe vraag aan de fabrikant wees me echter in de richting van Newtech Component Ltd., een distributeur die zo vriendelijk was om 20 monsters gratis op te sturen. Hiervoor is het raadzaam om gebruik te maken van een vracht- of pakketdienst die een adres kan leveren (in de buurt van de distributeur) waar de monsters naartoe kunnen worden gestuurd. De auteur gebruikt hiervoor vaak TipTrans.

Het board wordt geleverd met compleet geïnstalleerde standaard-firmware, waardoor eventuele 'hobbels' bij de installatieprocedure van het systeem vermeden worden. Bij het ESP32-S3-EYE-board met fabrieksinstellingen hoef je alleen maar een voeding via de micro-USB-poort aan te sluiten en een paar seconden te wachten. De gezichtsherkennings-applicatie wordt meteen gestart, zoals te zien in **figuur 2**. Het is de moeite waard om hier op te merken dat AI- en ML-algoritmen opmerkelijk robuust zijn met betrekking tot de kwaliteit van de opgenomen beelden. De foto in figuur 2 is gemaakt met de beschermende plasticfolie nog over de cameralens. Het verminderde contrast (en mijn ongeschoren gezicht) konden het AI-gezichtsherkennings-proces niet voor de gek te houden.

Back to the Future

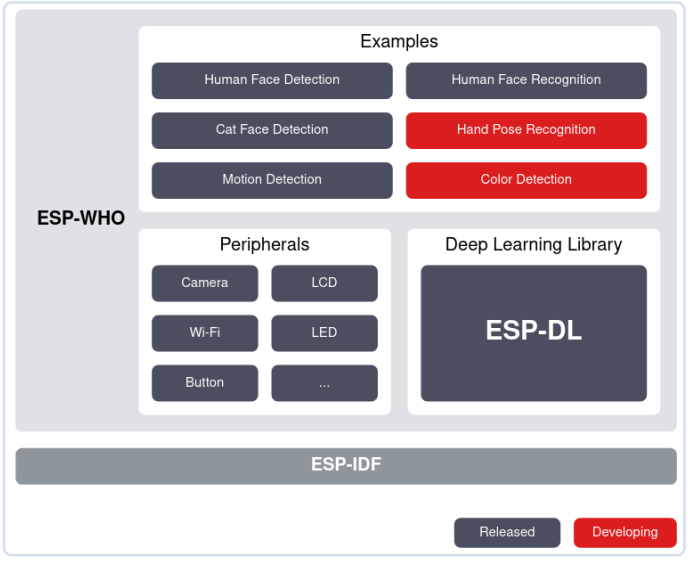
Misschien wil je je ESP32-S3-EYE op een bepaald moment terugzetten naar de 'fabrieksinstellingen.' Kant-en-klare .bin-bestanden zijn te vinden op de URL https://github.com/espressif/esp-who/tree/master/default_bin. Deze kunnen op dezelfde manier naar het board worden geflasht als elk ander binair bestand.



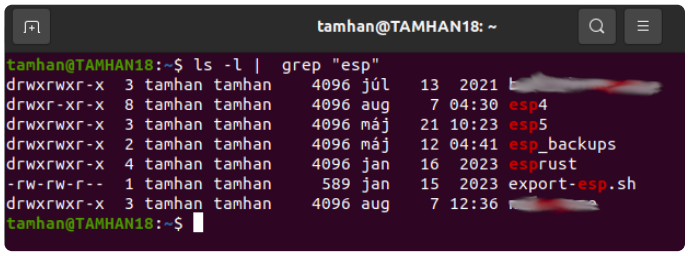
Figuur 2. De camera werkte, zelfs in mijn rommelige lab met de lenstape er nog op!

Het eerste experiment

Toegegeven, de bijna onbeperkte beschikbaarheid van durfkapitaal heeft geleid tot een stortvloed aan AI-frameworks. Het is logisch dat die bedrijven zich niet alleen op PC's richten, maar ook microcontrollers in hun portfolio van ondersteunde platforms opnemen.



Figuur 3. ESP-WHO maakt gebruik van verschillende andere onderdelen van het Espressif-ecosysteem (bron: [11]).



Figuur 4. Verschillende versies van de ESP-IDF kunnen over het algemeen naast elkaar op een werkstation bestaan.

Een uitgebreid overzicht van de huidige marktsituatie zou buiten het bestek van dit artikel vallen, dus voor onze eerste stap werken we met het ontwikkelplatform *ESP-WHO* voor beeldverwerking, beheerd door Espressif. Dit is een framework dat 'geoptimaliseerd' is voor bepaalde soorten herkenning van bewegende beelden, en de structurele opzet wordt getoond in **figuur 3**.

De *ESP Deep Learning Library* (ESP-DL), beschikbaar op [3], is een belangrijke bron. Dit is in wezen een geoptimaliseerde bibliotheek die verschillende AI-technieken biedt en een aanzienlijke latency-reductie bereikt wanneer deze wordt gebruikt in combinatie met een hardware-versneller.

Een punt van aandacht is dat de WHO-component standaard alleen draait op versie 4.4 van de ESP-IDF en nog niet wordt ondersteund in versie 5.0. Voor mijn consultancywerk gebruik ik over het algemeen versie 4.4 voor projecten, maar ik heb meerdere varianten van de ESP32-ontwikkelomgeving op mijn computer geïnstalleerd, zoals je kunt zien in **figuur 4**.

De IDF-versie die in de volgende stappen wordt gebruikt, wordt als volgt geïdentificeerd:

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

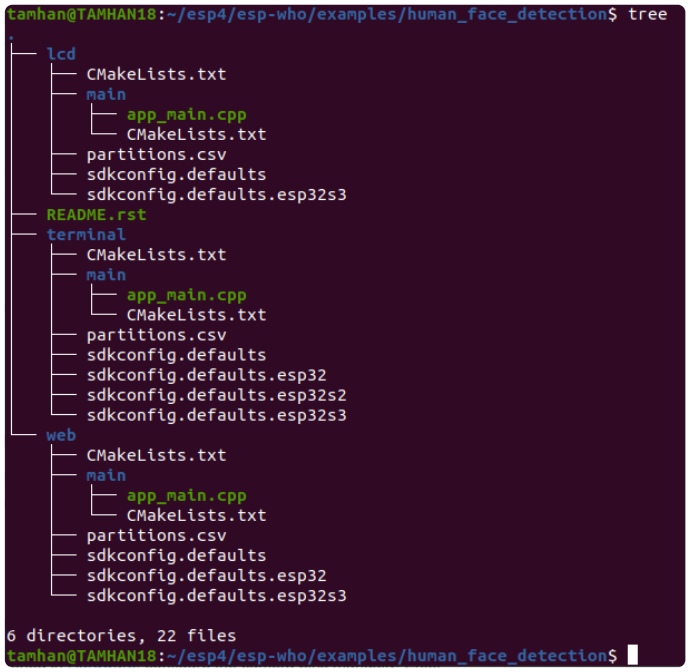
We zijn nu klaar om de *ESP-WHO* code te downloaden. Dit wordt gedaan met dit `git`-commando in de commandoregel:

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

De stap `Compressing objects:` kan enige tijd in beslag nemen; repositories zijn soms vrij groot en traag om mee te werken. Nadat het werk voltooid is, is het aan te raden om de submodule op de volgende manier te initialiseren:

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

In de meeste gevallen zal het commando `git submodule update --recursive --init` geen informatie op het scherm tonen – en dat betekent dat het huidige image 'compleet' is.



Figuur 5. ESP-IDF interne varianten.

Een paar voorbeelden

De gezichtsherkenningssoftware *ESP-WHO* van Espressif biedt drie verschillende manieren voor de gebruikersinterface. Deze voorbeeldprojecten zijn te zien in **figuur 5**.

In de volgende stappen gebruiken we de LCD-variant van de voorbeelden `~/esp4/esp-who/examples/human_face_detection`. De berekende resultaten worden dan direct weergegeven op het kleine schermpje dat op de ESP32 is gemonteerd. De terminalvariant gebruikt de *idf.py* monitor voor de communicatie, terwijl de *Web*-variant een webserver opzet.

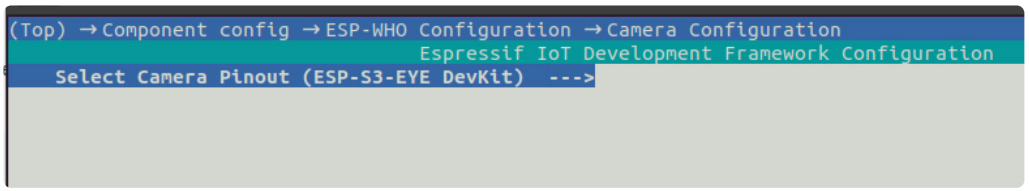
Eerst moet je het type ESP32-controller specificeren waarop je je wilt richten, met behulp van het volgende schema. Als je net als ik een ESP32-S3-EYE board gebruikt, wordt de string `esp32s3` doorgegeven met behulp van:

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

De eigenlijke broncode van het voorbeeld, die zich bevindt in het bestand *main/app_main.cpp*, is bijzonder eenvoudig. Om een beter inzicht te krijgen, zal ik het eerst in zijn geheel afdrukken voordat ik commentaar toevoeg (zie **listing 1**).

De kern van de *ESP-WHO* engine gebruikt het *Queue* kernel-object dat is geïmplementeerd in FreeRTOS, zoals beschreven in [4]. De twee statische defenities leveren een *Input* en een *Output Queue*, die later worden gebruikt om de datastroom door het herkenningsproces af te handelen.

De rest van de code die de ontwikkelaar heeft geschreven is voornamelijk gericht op het bouwen van een pijplijn via aanroepen van drie functies. Er zijn duidelijk overeenkomsten met het pijplijnmodel dat wordt gebruikt door het ESP-ADF audio framework [5].



Figuur 6. Als je een interne cameramodule gebruikt, moet je die hier opnieuw configureren!

Listing 1. In de wachtrij.

```
#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);
}
```

Om verder te gaan met de installatie, moet je in de eerste stap `idf.py menuconfig` openen om de bekende *menuconfig* configuratieomgeving te laden die in andere ESP32-projecten (en de Linux-kernel) wordt gebruikt. Navigeer daarna naar *Component config* ➜ *ESP-WHO Configuration*. Zorg er in het veld *Camera Configuration* voor dat de camera van je evaluatieboard is voorgeconfigureerd zoals in **figuur 6**. Sla de zo gemaakte build-configuratie in *menuconfig* op en voer dan het gebruikelijke `idf.py build` commando uit om het compileren te starten. De aanvankelijke aanmaak van de image neemt wat meer tijd in beslag, omdat het framework ongeveer 1200 codebestanden moet compileren.
Voordat je het board flasht met `idf.py flash` en het poortnummer, moet je het board in bootloader-modus zetten. Gebruik de twee knoppen bij de USB-connector: houd BOOT ingedrukt terwijl je kort op RST drukt en laat dan beide knoppen los. Nu kun je zoals gebruikelijk het programma in de ESP32 flashen. Je kunt de toegang tot de bootloader-modus herkennen in *dmesg*, zoals te zien in **figuur 7**. Druk nu nogmaals op de RST-knop om de nieuwe variant van de bekende gezichtsherkenning (figuur 2) te starten.

Een snelle blik op de code

Het gebruik van *ESP-WHO* biedt een relatief eenvoudige introductie voor ontwikkelaars om zonder al te veel moeite te beginnen met het experimenteren met AI-toepassingen. Espressif levert ook de broncode voor sommige componenten [6], waardoor we de gezichtsherkenningsroutine kunnen bekijken.
Het gebruikt de al bestaande `HumanFaceDetectMSR01` detector, die enkele gewichten krijgt doorgegeven als onderdeel van de parametreer-procedure:

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

Het eigenlijke werk wordt gedaan door het ML-model in een eindeloze lus, die individuele frames ophaalt uit de wachtrij die eerder is aangemaakt:

```
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});
```

Het aanroepen van de functie `detector.infer()` zorgt ervoor dat het eigenlijke inferentieproces wordt uitgevoerd. Als het geretourneerde `results`-object iets gedetecteerd heeft, roept het programma twee functies aan voor de uitvoer:

```
    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:~$
```

Figuur 7. Dit statusbericht geeft een succesvolle verbinding aan.

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

Figuur 8. Door de AI-versneller te activeren wordt de klus sneller geklaard (bron: [9]).

De rest van *ESP-WHO* bestaat grotendeels uit bestaande softwarecomponenten zoals *ESP-Cam* [7], om de toegang tot de camera af te handelen. Ik zal niet verder ingaan op de *ESP-WHO* software omdat de gegeven voorbeelden voor zich spreken.

Dieper gaan met TensorFlow

Wie meer controle wil over het gedrag van zijn systeem, kan in de verleiding komen om helemaal vanaf nul te beginnen met coderen, maar dat is geen bijzonder praktische oplossing. Tenzij je er veel zorg aan besteedt, kan het systeem snel onbeheersbaar worden en leiden tot hogere onderhoudskosten gedurende de levensduur van de software. De *TensorFlow*-bibliotheek van Google, beschikbaar op [8], is een soort quasi-standaard geworden en is al enige tijd beschikbaar in geoptimaliseerde versies voor verschillende microcontrollers. Het is belangrijk om op te merken dat het op GitHub eigenlijk uit twee repositories bestaat: de reden voor deze eigenaardigheid is dat Google de distributie van algemene en leverancierssspecifieke code in de *TensorFlow*-bibliotheek halverwege de ontwikkeling van het framework heeft verschoven. De versie van de bibliotheek die we nodig hebben is te vinden op [9].
Omdat TensorFlow ook toegang heeft tot verschillende componenten van het ESP-IDF framework op de achtergrond, moet de implementatie vanaf GitHub ook plaatsvinden met de `--recursive` parameter die de opdrachtregel-tool wijst op de noodzaak om gerelateerde codelocaties aan te leveren en ervoor zorgt dat we de hoofdrepository en alle submodules krijgen:

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

Voor een eerste praktijktest kunnen we het voorbeeld *~/esp4/tflite-micro-esp-examples/examples/hello_world* gebruiken. Het gebruikt een ML-model dat is geoptimaliseerd door sinusfunctiewaarden te 'voorspellen', een alternatief voor het meer conventionele CORDIC-algoritme en een model dat wordt uitgevoerd met minimale invoergegevens. Voer in de volgende stap weer `idf.py set-target esp32s3` in om het projectmodel te optimaliseren voor het ESP32-S3 target.
Afhankelijk van de versie van ESP-IDF die beschikbaar is op je werkstation of PC, kun je foutmeldingen tegenkomen tijdens het instellen van de parameters van de code die is gedownload uit de repository (*Invalid manifest…*). Deze fouten wijzen op gedeeltelijk verouderde componenten in de compilatie-toolchain. Om dit op te lossen kun je het volgende programma uitvoeren via de commandoregel:

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

Open vervolgens een *Nautilus*-venster dat naar de hoofdmap van het project-'skelet' wijst door het volgende commando in te voeren. Het wissen van de *build*-map moet handmatig worden gedaan, zodat het `set-target` commando de ondersteunende bestanden voor de compilatie opnieuw kan aanmaken:

```
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
```

Let op de *ESP-NN* ingang in *Menuconfig*; deze staat het configureren van de DL bibliotheek toe. Het selecteren van de *Optimized* versions optie in de sectie *Optimization for neural network functions* is zeer aan te raden omdat dit de versneller activeert. De prestatiecijfers in **figuur 8** laten de aanzienlijke toename zien die kan worden verwacht als deze optie is ingeschakeld.
Het compileren en uitvoeren verloopt dan zoals verwacht. In **figuur 9** zie je de uitvoer van de berekende resultaten.

```
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
```

Figuur 9. Neurale netwerken werken ook met sinussen.

Analyse van de TensorFlow-code

Wees niet verbaasd als je chaotische beelden ziet op het ESP-EYE display – het ingebouwde LCD heeft zijn eigen framebuffer-controller, die gewoon het laatst opgeslagen beeld weergeeft totdat er een nieuwe update is.
Als je het bestand *main.cc* opent in een teksteditor naar keuze, vind je het volgende fragment:

```
#include "main_functions.h"

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

Eerste indrukken zijn hier niet misleidend. TensorFlow sluit nauw aan bij de Arduino-omgeving. De feitelijke implementatie van de `setup()` en `loop()` functies is te vinden in het *main_functions.cc* bestand.
De `loop()`-methode is vooral interessant omdat deze verantwoordelijk is voor het uitvoeren van de payloads. De eerste taak is het genereren van invoergegevens die worden ingevoerd in de sinus-voorspeller:

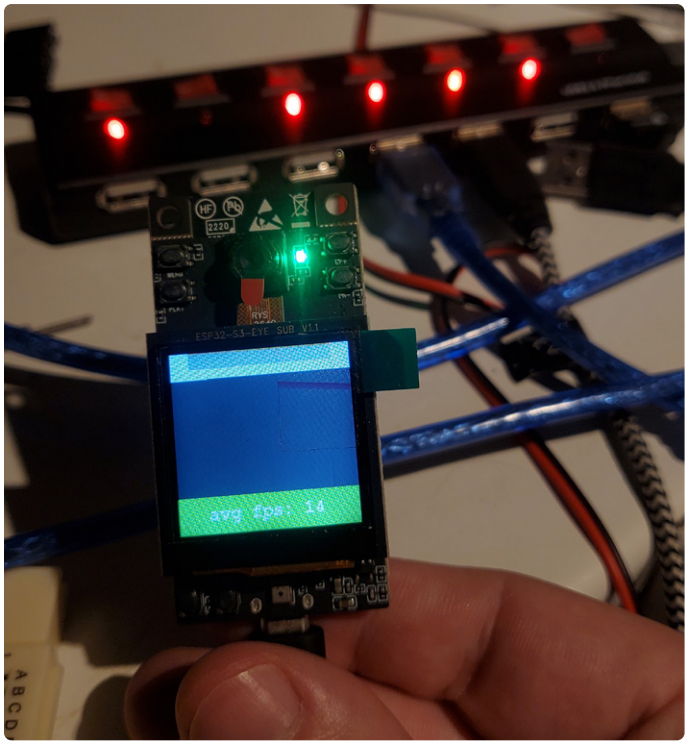
```
void loop() {
  float position =
      static_cast<float>(inference_count) /
      static_cast<float>(kInferencesPerCycle);
  float x = position * kXrange;
```

In de volgende stap wordt de informatie gekwantificeerd om deze beter 'verteerbaar' te maken voor het neurale netwerk. Dit is een populaire aanpak op het gebied van machine learning. Het normaliseren van alle invoerwaarden naar een bereik tussen 0 en 1 helpt om de complexiteit van het resulterende algoritme binnen de perken te houden:

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

De eigenlijke berekening en kwantificering vinden dan plaats in het volgende blok:

```
  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;
```



Figuur 10. De groene footer geeft een succesvolle herkenning aan.

Tot slot is er nog wat huishoudelijk werk nodig. Het is interessant om op te merken dat volgens de TensorFlow Micro-documentatie, de functie die verantwoordelijk is voor de daadwerkelijke gegevensuitvoer, `HandleOutput()`, moet worden geleverd door de ontwikkelaar:

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

Experimenten met meer geavanceerde modules

Als je de map met voorbeelden *~/esp4/tflite-micro-esp-examples/examples* van Google zorgvuldig analyseert, zul je zien dat er een spraakherkenningsvoorbeeld in zit met de naam *micro_speech* en zelfs een gezichtsherkenningsvoorbeeld met de naam *person_detection*.
Om deze voorbeelden aan de praat te krijgen, moet je de bekende drietraps-procedure volgen van parameters instellen, *menuconfig*-instellingen controleren, compileren, gevolgd door het uploaden van de machinecode naar de ESP32.
Het is belangrijk op te merken dat deze geavanceerde voorbeelden meestal werken met behulp van commandoregel-communicatie. Als je schermuitvoer wilt toevoegen aan de personendetector, moet je het bestand *esp_main.h* als volgt aanpassen:

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

Na het hercompileren verschijnt het scherm van **figuur 10**. Het oplossen van het weergaveprobleem is het onderwerp van een ander artikel. Als de weergegeven voettekst groen wordt, betekent dit dat de herkenning geslaagd is.

Twee methodes

De experimenten die hier zijn uitgevoerd laten zien dat het ESP32-S3 platform in staat is om op verschillende manieren object- en gezichtsherkennings-payloads uit te voeren. Terwijl het ESP-WHO pad snelle en indrukwekkende resultaten oplevert, maakt de methode om het te integreren in het TensorFlow-ecosysteem het gebruik van geavanceerde technieken op het gebied van machinel learning mogelijk.

230556-03 (vertaling: Hans Adams)

Vragen of opmerkingen?

Hebt u vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via tamhan@tamoggemon.com of naar de redactie van Elektor via redactie@elektor.com.

Gerelateerd product

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

WEBLINKS

[1] ESP32-S3-EYE distributeur zoeken: https://oemsecrets.com/compare/ESP32-S3-EYE
[2] Schema van de ESP32-S3-EYE: https://tinyurl.com/esp32eyeschematics
[3] ESP Deep Learning-bibliotheek: https://github.com/espressif/esp-dl
[4] FreeRTOS Queues: https://freertos.org/a00018.html
[5] T. Hanna, "Audio signalen en de ESP32", Elektor januari/februari 2023: https://www.elektormagazine.nl/magazine/elektor-290/61475
[6] AI-voorbeeld ESP-WHO: https://github.com/espressif/esp-who/tree/master/components/modules/ai
[7] ESP32 camerabesturing: https://github.com/espressif/esp32-camera
[8] TensorFlow: https://tensorflow.org
[9] TensorFlow Lite Micro: https://github.com/espressif/tflite-micro-esp-examples
[10] ESP32-S3-EYE blokschema:
https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md
[11] ESP-WHO op GitHub: https://github.com/espressif/esp-who